COMP 3311 DATABASE MANAGEMENT SYSTEMS

TUTORIAL 8 QUERY PROCESSING / QUERY OPTIMIZATION

REVIEW: MATERIALIZATION AND PIPELINING

Materialization

– Generate results of an expression whose inputs are relations or are already computed and materialize (store) it on disk.

– Repeat.

Overall cost: Sum of costs of individual operations + cost of writing intermediate results to and reading intermediate results from disk.

Pipelining

– Pass on tuples to parent operations even as an operation is being executed.

– Avoids writing/reading intermediate results to/from disk.

Overall cost: Sum of costs of individual operations.

EXERCISE 1

Consider the relations R_1(<u>A</u>, B, *C*), R_2(<u>C</u>, D, *E*) and R_3(<u>E</u>, F). Primary keys are underlined and foreign keys are in italics.

Assume that:

R_1 has 1,000 tuples in 100 pages.

R_2 has 10,000 tuples in 1,000 pages.

R_3 has 100,000 tuples in 10,000 pages.

a) What is the size of the final result of $R_1 \bowtie R_2 \bowtie R_3$?

b) Give an efficient pipelining strategy to compute $R_1 \bowtie R_2 \bowtie R_3$.

R_1(A, B, C)	1,000 tuples	100 pages
R_2(C, D, E)	10,000 tuples	1,000 pages
R_3(E, F)	100,000 tuples	10,000 pages

EXERCISE 1 (CONT'D)

a) What is the size of the final result of $R_1 \bowtie R_2 \bowtie R_3$?

- Since all joins are on key values, the size of the final result is equal to the size of the smallest relation R_1, which is 1000 tuples.

☞ **$(R_1 \bowtie R_2) \bowtie R_3$ is less costly than $R_1 \bowtie (R_2 \bowtie R_3)$. Why?**

- $R_1 \bowtie R_2$ will generate 1,000 tuples; then the join with R_3 will also generate 1,000 tuples.

- $R_2 \bowtie R_3$ will generate 10,000 tuples; then the join with R_1 will generate 1,000 tuples.

- Even though the final result is the same, the computation for $(R_1 \bowtie R_2) \bowtie R_3$ will be less costly since fewer total tuples are generated.

R_1(A, B, C)	1,000 tuples	100 pages
R_2(C, D, E)	10,000 tuples	1,000 pages
R_3(E, F)	100,000 tuples	10,000 pages

EXERCISE 1 (CONT'D)

b) Give an efficient pipelining strategy to compute $(R_1 \bowtie R_2) \bowtie R_3$.

- create an index on attribute C for relation R_2.
- create an index on attribute E for relation R_3.

As the condition is equality on a key attribute, a hash index is best.

Then, for each tuple in R_1, we do the following:

a. Search in R_2 to find the match with the C value of R_1. **Cost:** 2 page I/Os
   1 page I/O to find the entry C in the hash index (assuming no overflow buckets).
   1 page I/O to retrieve the tuple from the relation R_2.

b. Then, for the result tuple of $(R_1 \bowtie R_2)$, search in R_3 to look up at most 1 tuple which matches the unique value for E in R_2. **Cost:** 2 page I/Os
   Again, we need 1 page I/O to access the index and 1 page I/O to retrieve the tuple.

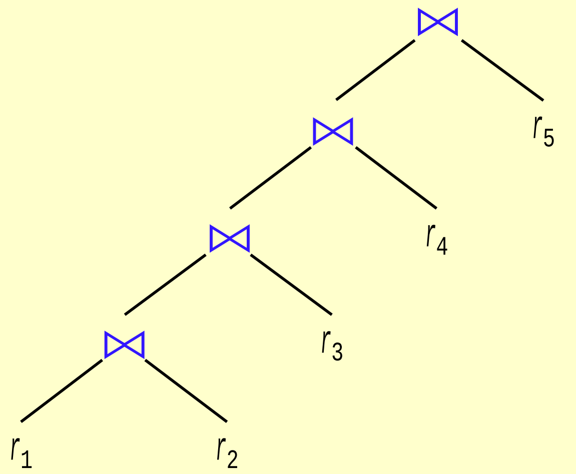
Total cost: 1000 * (2 + 2) + 100 = 4100 page I/Os
(where 100 is the cost for reading the 1,000 tuples of R_1)

This plan corresponds to an indexed nested-loop join and ignores the cost of building the indexes.

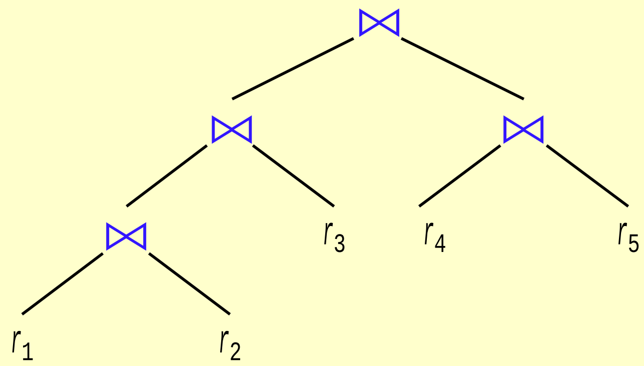
REVIEW: QUERY OPTIMIZATION

Heuristic Optimization

- Perform selections early.
- Perform projections early.
- Perform most restrictive selections and join operations before other similar operations ⟹ create left-deep join trees.



Left-deep join tree



Non-left-deep join tree

EXERCISE 2

Student(studentId, sName, gender) — 1,000 tuples; 100 pages; index on studentId

EnrollsIn(studentId, courseId, year) — 6,000 tuples; 600 pages; index on courseId

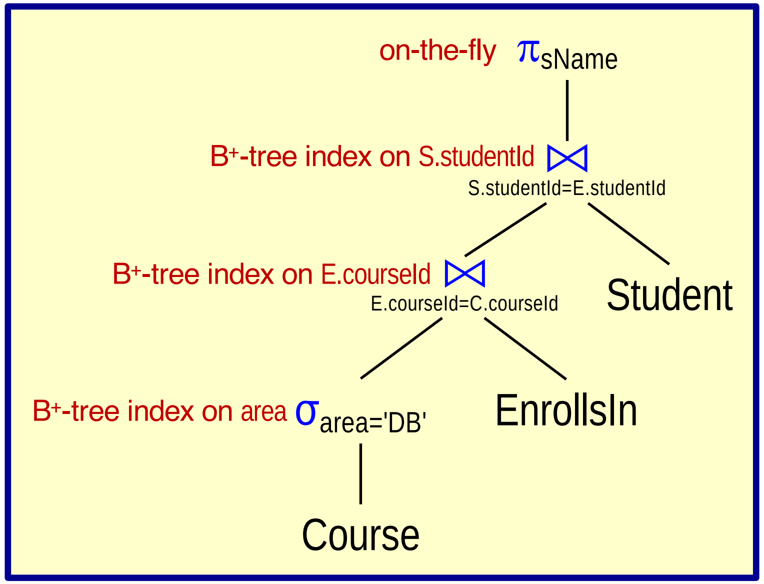
Course(courseId, cName, area, credit) — 200 tuples; 40 pages; index on area
10 different areas, with 20 tuples per area

All indexes are B+-tree clustered indexes with 4 levels.

The EnrollsIn tuples are uniformly distributed among students and courses.

Using a pipelined plan to process the relational algebra tree for the query below, answer the following questions.

```
select sName
from Student S, EnrollsIn E, Course C
where S.studentId=E.studentId
      and E.courseId=C.courseId
      and area= 'DB';
```

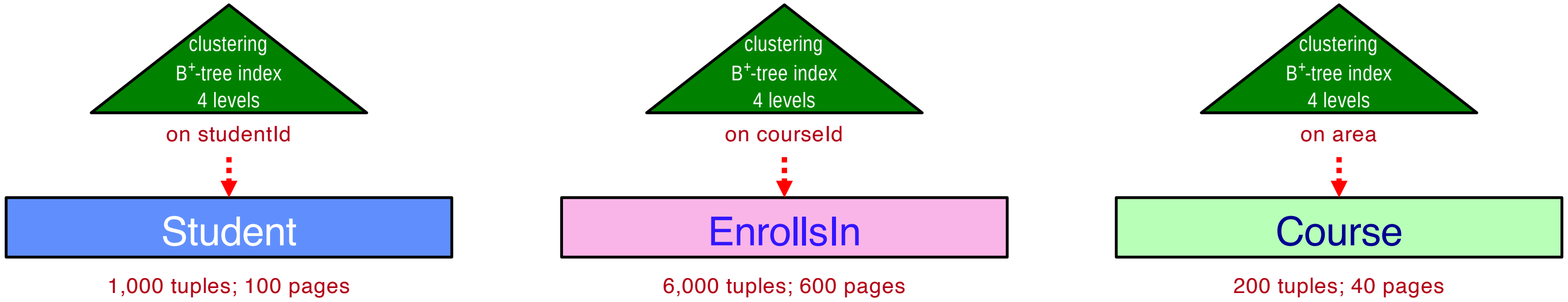


EXERCISE 2 (CONT'D)

Student(<u>studentId</u>, sName, gender)

EnrollsIn(<u>*studentId, courseId*</u>, year)

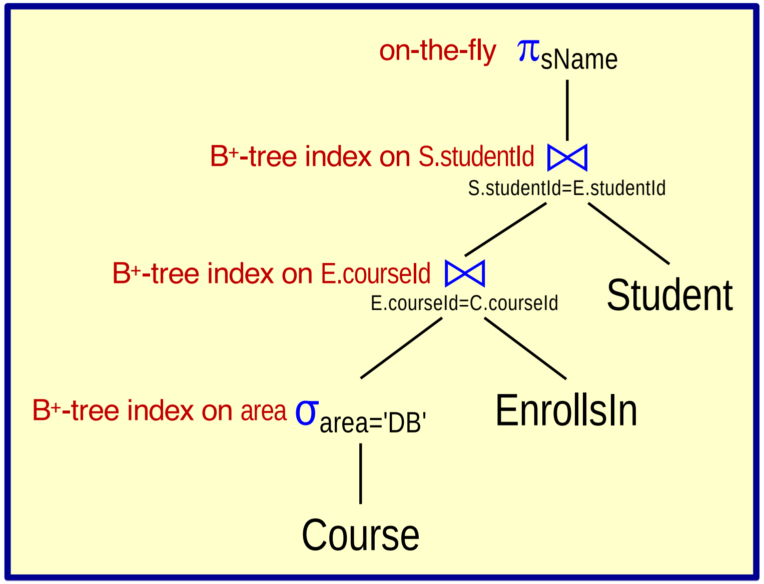
Course(<u>courseId</u>, cName, area, credit)




```
select sName
from Student S, EnrollsIn E, Course C
where S.studentId=E.studentId
      and E.courseId=C.courseId
      and area= 'DB';
```

<u>Some useful statistics:</u>

- 10 different areas.
- 20 tuples per area.



Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 2 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

a) Estimate the result size of the query.

$\pi_{sName}(\sigma_{area='DB'}(\text{Student} \bowtie_{studentId} \text{EnrollsIn} \bowtie_{courseId} \text{Course}))$

EnrollsIn $\bowtie_{courseId}$ Course: courseId is a foreign key referencing Course in EnrollsIn. Thus, the size of (EnrollsIn $\bowtie_{courseId}$ Course) is the same as the size of EnrollsIn (i.e., 6000 tuples).

Student $\bowtie_{studentId}$ EnrollsIn $\bowtie_{courseId}$ Course: studentId is a foreign key referencing Student in (EnrollsIn $\bowtie_{courseId}$ Course). Thus, the size of (Student $\bowtie_{studentId}$ (EnrollsIn $\bowtie_{courseId}$ Course)) is the same as the size of (EnrollsIn $\bowtie_{courseId}$ Course), which is the same as the size of EnrollsIn (i.e., 6000 tuples).

$\sigma_{area='DB'}$(Student $\bowtie_{studentId}$ EnrollsIn $\bowtie_{courseId}$ Course): The selectivity of the condition area='DB' is 1/10 (i.e., 10 different areas). Thus, the size of $\sigma_{area='DB'}$(Student $\bowtie_{studentId}$ EnrollsIn $\bowtie_{courseId}$ Course) is 6000/10=600 tuples.

Estimated output size: 600 tuples

Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 2 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

b) Estimate the cost for this query using the pipelined plan.

Step 1: $\sigma_{area='DB'}$Course $\Longrightarrow$ result A

Strategy: index lookup using B$^+$-tree on area

V(Course, area) = 10 distinct area values.

Thus, there are 200/10 = 20 Course tuples (or 40/10 = 4 Course pages) whose area='DB'. Note that Course must be ordered on area due to the clustering index on area.

☞ There are 20 distinct course ids for area='DB'.

To get all these Course tuples requires

4 index page I/Os, since each B$^+$-tree index has 4 levels.

4 Course page I/Os, since Course is ordered on area.

Cost: 8 page I/Os

Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 2 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

Step 2: result A ⋈ EnrollsIn ⟹ result B

Strategy: indexed nested-loop join using courseId B^+-tree index

For each courseId selected from Course, we use the clustering index on courseId for EnrollsIn to locate all related EnrollsIn tuples and get the corresponding studentId.

V(Course, courseId) = 200 distinct courseId values.

We assume that courses are uniformly distributed among EnrollsIn tuples. Thus, for each courseId value there are 6000/200 = 30 tuples (or 600/200 = 3 pages) in EnrollsIn. Note that EnrollsIn must be ordered on courseId due to the clustering index on courseId.

To get all the EnrollsIn tuples *for each courseId value* requires

- 4 index page I/Os, since each B^+-tree index has 4 levels.
- 3 EnrollsIn page I/Os, since EnrollsIn is ordered on courseId.
- **Cost per courseId:** 7 page I/Os

There are 20 distinct courseId values for area='DB'.

Cost: 20 * 7 = 140 page I/Os

Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 2 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

Step 3: result B ⋈ Student

Strategy: indexed nested-loop join using studentId B^+-tree index

For each studentId selected from EnrollsIn, we use the clustering index to locate the related tuple (in Student) and get the corresponding sName.

To get the Student tuple *for each studentId value* requires

- 4 index page I/Os, since each B^+-tree index has 4 levels.
- 1 Student page I/O, to get the Student tuple.

Cost per studentId: 5 page I/Os

There are 20 courseId values and 30 EnrollsIn tuples for each courseId value, Thus, there are 20*30 = 600 studentId values.

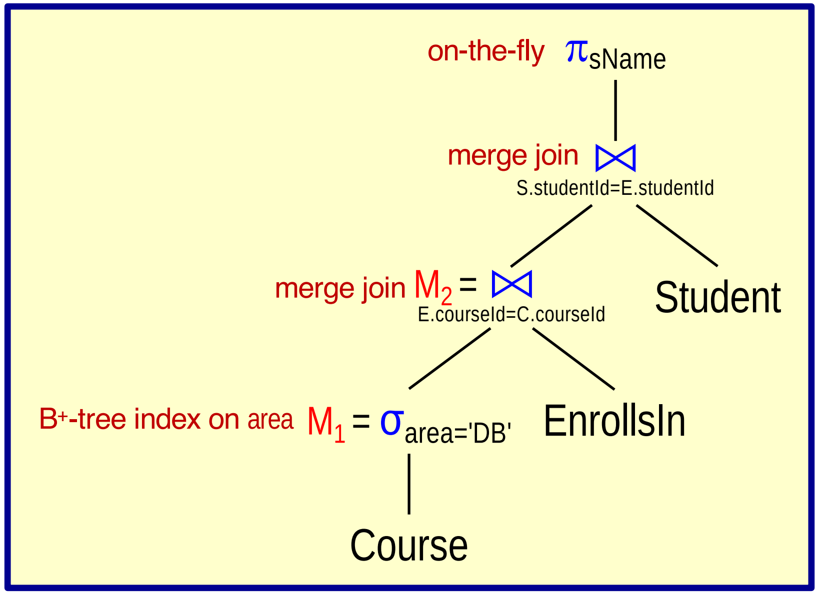
Cost: 600 * 5 = 3000 page I/Os

Total cost: 8 + 140 + 3000 = 3148 page I/Os

EXERCISE 3

Suppose you are allowed to materialize the intermediate results $M_1 = \sigma_{area='DB'}$ and $M_2 = M_1 \bowtie_{courseId}$ EnrollsIn for the relational algebra tree shown below. What is the query processing cost assuming that merge join is used for all joins?

Assume there are 22 pages of main memory and attributes in the same relation all have the same size.



Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 3 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

a) Cost to materialize $M_1 = \sigma_{area='DB'}$

Strategy: index lookup using B^+-tree on area

From the previous analysis we know that to evaluate $\sigma_{area='DB'}$ requires 8 page I/Os and results in 4 pages of Course tuples.

Since we have 22 pages of memory, we can sort these 4 pages of Course tuples on courseId in memory and materialize the result (i.e., write it to disk) as M_1.

We need to keep only courseId in the result since only the coursed is needed to join with EnrollsIn.

Since we assume that all attributes in the same relation have the same size and there are 4 attributes in Course, the size of the output is 4 pages/4 = 1 page.

Cost to materialize M_1: $\overset{r}{8} + \overset{w}{1} = \underline{9}$ page I/Os

Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 3 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

b) Cost to materialize M_2=$M_1 \bowtie_{courseId}$EnrollsIn

Strategy: merge join

The merge join cost is 1 page I/O to read M_1 and 600 page I/Os to read EnrollsIn. (EnrollsIn is sorted on coursed due to B+-tree clustering index.)

We keep only studentId since only studentId is needed to join with Student.

From the previous analysis we know that for each courseId value we access 3 pages. Since we only keep the studentId in the result and EnrollsIn has 3 attributes, the size of the result is 3 pages/3 = 1 page for each courseId value. Since there are 20 distinct courseId values, the result size is 1*20 = 20 pages.

With 22 pages of main memory, 2 pages can be used for reading M_1 and EnrollsIn and the remaining 20 pages can hold the join result. Note that since EnrollsIn is clustered on courseId, there is no need to sort it.

We next do an in-memory sort of the result (on studentId) and materialize it as M_2 requiring 20 page I/Os to write M_2 to disk.

Cost to materialize M_2: $\overset{r}{1} + \overset{r}{600} + \overset{w}{20}$ = 621 page I/Os

Student(studentId, sName, gender)
EnrollsIn(studentId, courseId, year)
Course(courseId, cName, area, credit)

EXERCISE 3 (CONT'D)

Student:	1000 tuples;	100 pages;	index on studentId
EnrollsIn:	6000 tuples;	600 pages;	index on courseId
Course:	200 tuples;	40 pages;	index on area
10 different areas		20 tuples per area	

c) Cost to compute $M_2 \bowtie_{studentId}$ Student

Strategy: merge join

20 page I/Os are required to read M_2 and 100 page I/Os to read Student.

Student has a clustering B⁺-tree index on studentId, Therefore, the Student relation is clustered on studentId and there is no need to sort it.

Cost to join M_2 and Student: 20 + 100 = 120 page I/Os

Total cost: 9 + 621 + 120 = 750 page I/Os