# Discussion 06/01

## Discussion 12-6

Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties.

- $r_1$ : 20,000 tuples; 25 tuples/block
- $r_2$ : 45,000 tuples; 20 tuples/block
- Buffer size: 64 blocks

Estimate the number of block transfers required for $r_1 \bowtie r_2$ using *block nested-loop join*.

r1 의 block 수는 800, r2 의 block 수는 2250. 버퍼 수가 64 개이므로 Cost 를 계산하면 [800 / 62] * 2250 + 800 혹은 [2250 / 62] * 800 + 2250. 두 가지 경우 중에 더 작은 걸 택하면 될 듯.

## Discussion 12-7

Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties.

- $r_1$ : 20,000 tuples; 25 tuples/block
- $r_2$ : 45,000 tuples; 20 tuples/block
- Buffer size: 64 blocks

Estimate the number of block transfers required for $r_1 \bowtie r_2$ using *hash join*.

3 * [800 + 2250] + 4n_h 인데 n_h 는 파티션을 몇 개로 하느냐에 따라 다름.

## Discussion 12-8

Perform a *hash join* of *r* and *s* on the numeric attribute. Assume $M = 4$ and use modulo as your hash function.

relation *r*

| | |
|---|---|
| 3 | A |
| 10 | B |
| 8 | C |
| 6 | D |
| 1 | E |
| 9 | F |
| 8 | G |
| 2 | H |
| 7 | I |
| 2 | J |
| 4 | K |
| 5 | L |
| 3 | M |
| 6 | N |
| 9 | O |
| 1 | P |

| | |
|---|---|
| 6 | b |
| 1 | a |
| 3 | c |
| 2 | e |
| 5 | f |
| 4 | g |
| 9 | k |
| 7 | n |

relation *s*

=〉 M 은 보통 꽤 큼. Partition 개수보다 일반적으로 memory buffer 가 더 큼. 적절한 partition 수는 어떻게 정하는 것이 좋을까? 작은 table 의 partition 된 각 block 들이 메인 메모리 buffer 에 완전히 들어올 수 있게 만들어 주는 것이 좋음. 따라서 각 파티션이 2 개였으면 좋음.

## Discussion 12-9

What would be the most efficient way to evaluate the following query? Represent your (heuristic) answer in a query evaluation tree.

**SELECT** *i_id, c_id, title*
**FROM** *teaches, course*
**WHERE** *teaches.c_id = course.c_id*

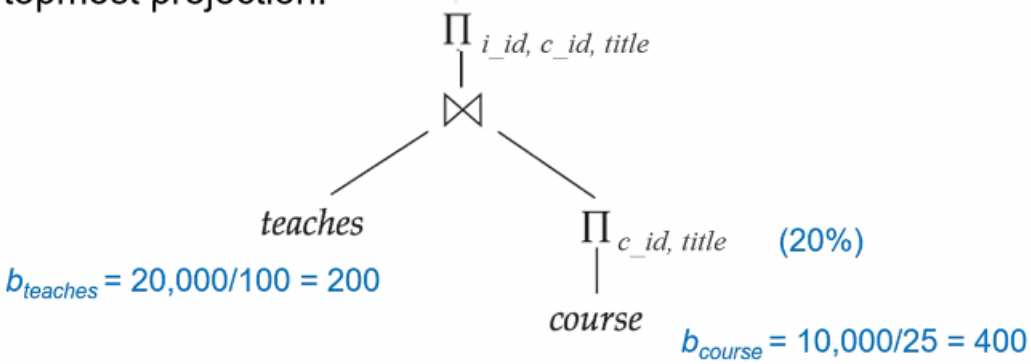
- ***teaches(i_id, c_id, year, semester)***
  $n_{teaches}$ = 20,000; $b_{teaches}$ = 200; *size(i_id, c_id)* = 50%
- ***course(c_id, title, dept, credit, level, description)***
  $n_{course}$ = 10,000; $b_{course}$ = 400; *size(c_id, title)*=20%
- ***M***=22 (# of buffer blocks)

제일 먼저 두 개의 테이블 각각에 대해서 projection operation 을 먼저 수행하면, size 가 teaches 는 50%, course 는 20%로 줄어든 상태에서 join 을 하기 때문에 효율적일 것 같고, 그 join 연산의 결과는 pipelining 을 해서 그대로 사용해서 위의 select operation 을 수행하면 좋을 것 같다. 이 때 c_id 를 알아야 하기 때문에 두 relation 모두에서 c_id 를 뽑았음.

즉, 최하위에 project_(i_id, c_id)_(teaches) 와 project_(c_id, title)_(course)가 있고, 걔네 두개가 join 으로 합쳐지고, 그 위에 select_(teaches.c_id = course.c_id)_(join 의 결과) 이렇게 나올 것 같음.

## Discussion 12-10

Suppose we chose the following evaluation plan for the previous query. Estimate the cost when we use materialization for all of the operations except for the topmost projection.



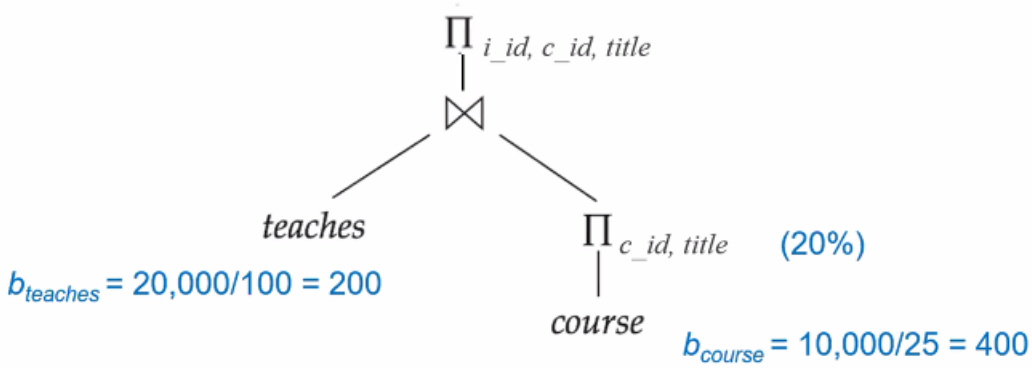

course 에서 projection 할 때 400 블락 읽고 80 블락 저장.

그럼 teaches 와 join 하면, 간단하게 hash 를 쓰면 3*(200+80) = 280*3 = 840 block transfer.

Join 후의 block 개수는 200 + 80*2 = 360 개.

## Discussion 12-11

Can you improve performance by *pipelining*? Which pairs of operations would you pipeline, and how much would you gain?



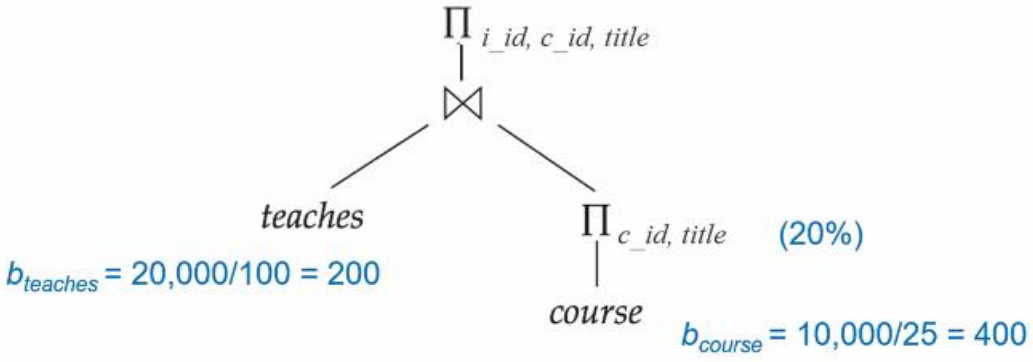

Join 후 projection 할 때 pipelining 이 가능. 또 course 에서 selection 을 한 결과를 pipelining 하면 80 개의 block 을 쓰지 않아도 되지 않을까?


=〉 근데 M 이 22 이므로 이런 식으로 20 개씩 나눠서 읽어들여서 그 때마다 teaches 를 읽는 식으로 해야 함.

그럼 join cost 가 [80/20] * 400 해서 800 임. 따라서 전체 cost 가 1200.

## Discussion 12-12

Suppose relations *teaches* and *course* are each sorted on the join attribute *c_id*. Would you use a different join algorithm? Justify.




***M***=22 (# of buffer blocks)

=〉 두 개의 relation 에 cursor 를 하나씩 두고 two pointer algorithm 을 사용해서 MERGE JOIN 을 함. Potentially 200~400 개의 block transfer. Merge join 은 join attribute 에 대해 sort 가 되어 있어야 함. Duplicate value 에 대해 처리를 해줘야 해서 Buffer 크기를 크게 잡음. 그래서 교수님이 반반으로 계산.