CSE 544:
Physical Operators

Monday, 5/17/2004

Question in Class

Logical operator:
Product(pname, cname) |×| Company(cname, city)

Propose three physical operators for the join, assuming the tables are in main memory:
1.
2.
3.

Question in Class

Product(pname, cname) |×| Company(cname, city)

- 1000000 products
- 1000 companies

How many comparisons do the following physical operators take if the data is **<u>in main memory</u>** ?

- Nested loop join comparisons =
- Sort and merge = merge-join comparisons =
- Hash join comparisons =

Cost Parameters

In database systems the data is on *disks*, not in main memory

The *cost* of an operation = total number of I/Os
Cost parameters:

- B(R) = number of blocks for relation R
- T(R) = number of tuples in relation R
- V(R, a) = number of distinct values of attribute a

Cost Parameters

- *Clustered* table R:
  - Blocks consists only of records from this table
  - B(R) ≈ T(R) / blockSize
- *Unclustered* table R:
  - Its records are placed on blocks with other tables
  - When R is *unclustered*: B(R) ≈ T(R)

- When a is a key, V(R,a) = T(R)
- When a is not a key, V(R,a)

Cost

Cost of an operation =
number of disk I/Os needed to:
- read the operands
- compute the result

Cost of writing the result to disk is <u>*not included*</u> on the following slides

<u>*Question*</u>: the cost of sorting a table with B blocks ?
<u>*Answer*</u>:

Nested Loop Joins

- Tuple-based nested loop R |×| S

```
for each tuple r in R do
   for each tuple s in S do
      if r and s join then output (r,s)
```

- Cost: T(R) B(S) when S is clustered
- Cost: T(R) T(S) when S is unclustered

Nested Loop Joins

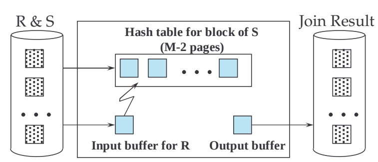
- We can be much more clever
- *Question*: how would you compute the join in the following cases ? What is the cost ?
  - B(R) = 1000, B(S) = 2, M = 4
  - B(R) = 1000, B(S) = 3, M = 4
  - B(R) = 1000, B(S) = 6, M = 4

Nested Loop Joins

- Block-based Nested Loop Join

```
for each (M-2) blocks bs of S do
   for each block br of R do
      for each tuple s in bs
         for each tuple r in br do
            if r and s join then output(r,s)
```

Nested Loop Joins



Nested Loop Joins

- Block-based Nested Loop Join
- Cost:
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-2) times, and each time need to read R: costs B(S)B(R)/(M-2)
  - Total cost: B(S) + B(S)B(R)/(M-2)
- Notice: it is better to iterate over the smaller relation first
- R |×| S: R=outer relation, S=inner relation

Merge-join

Join R |×| S

- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Difficulty: many tuples in R may match many in S
  - If at least one set of tuples fits in M, we are OK
  - Otherwise need nested loop, higher cost
- Total cost: 5B(R)+5B(S)
- Assumption: $B(R) <= M^2$, $B(S) <= M^2$

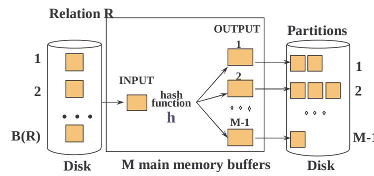
Merge-join

Join R |x| S

- If the number of tuples in R matching those in S is small (or vice versa) we can compute the join during the merge phase
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R) + B(S) <= M^2$

Partitioned Hash-based Algorithms

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if $B(R)/M <= M$, i.e. $B(R) <= M^2$

Hash Based Algorithms for δ

- Recall: $\delta(R)$ = duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption: $B(R) <= M^2$

Hash Based Algorithms for γ

- Recall: $\gamma(R)$ = grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply γ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption: $B(R) <= M^2$

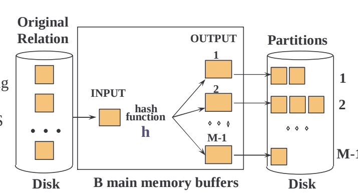
Partitioned Hash Join

R |x| S

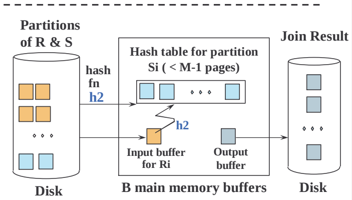
- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

Hash-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.



- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.



Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

Hybrid Hash Join Algorithm

- Partition S into k buckets
  t buckets S_1 , …, S_t stay in memory
  k-t buckets S_{t+1}, …, S_k to disk
- Partition R into k buckets
  – First t buckets join immediately with S
  – Rest k-t buckets go to disk
- Finally, join k-t pairs of buckets:
  (R_{t+1},S_{t+1}), (R_{t+2},S_{t+2}), …, (R_k,S_k)

Hybrid Join Algorithm

- How to choose k and t ?
  – Choose k large but s.t. k <= M
  – Choose t/k large but s.t. t/k * B(S) <= M
  – Moreover: t/k * B(S) + k-t <= M
- Assuming t/k * B(S) >> k-t: t/k = M/B(S)

Hybrid Join Algorithm

- How many I/Os ?
- Cost of partitioned hash join: 3B(R) + 3B(S)
- Hybrid join saves 2 I/Os for a t/k fraction of buckets
- Hybrid join saves 2t/k(B(R) + B(S)) I/Os
- Cost: (3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))

Indexed Based Algorithms

- Recall that in a clustered index all tuples with the same value of the key are clustered on as few blocks as possible

a a a	a a a a a	a a

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on a: cost= B(R)/V(R,a)
- Unclustered index on a: cost= T(R)/V(R,a)

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20, compute the cost of $\sigma_{a=v}(R)$
- Cost of table scan:
  - If R is clustered: B(R) = 2000 I/Os
  - If R is unclustered: T(R) = 100,000 I/Os
- Cost of index based selection:
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5000
- Notice: when V(R,a) is small, then unclustered index is useless

Index Based Join

- R |×| S
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Assume R is clustered. Cost:
  - If index is clustered: B(R) + T(R)B(S)/V(S,a)
  - If index is unclustered: B(R) + T(R)T(S)/V(S,a)

Index Based Join

- Assume both R and S have a sorted index (B+ tree) on the join attribute
- Then perform a merge join (called zig-zag join)
- Cost: B(R) + B(S)

Example

Product(<u>pname</u>, maker), **Company**(<u>cname</u>, city)

Select **Product**.pname
From **Product**, **Company**
Where **Product**.maker=**Company**.cname
and **Company**.city = "Seattle"

- How do we execute this query ?

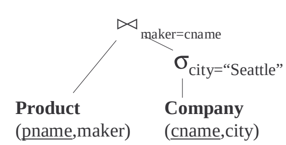
Example

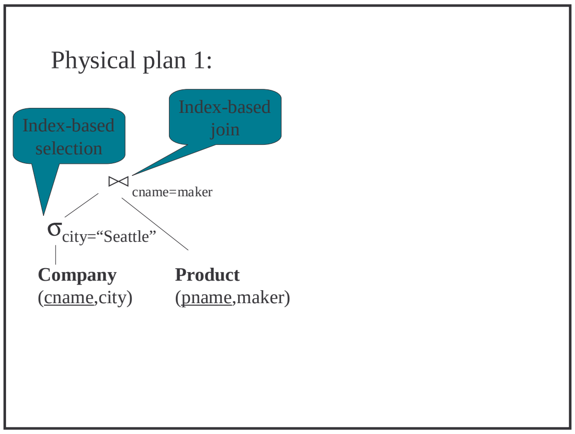
Product(<u>pname</u>, maker), **Company**(<u>cname</u>, city)

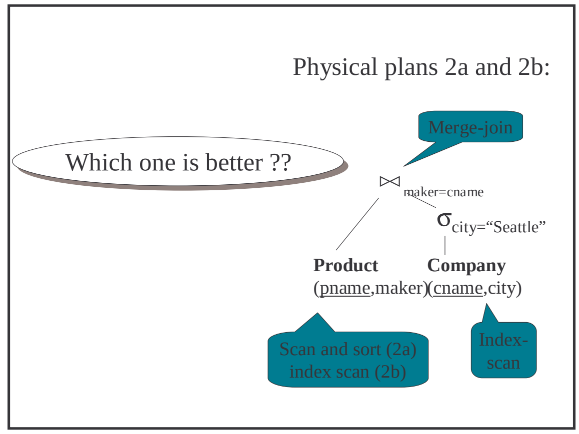
Assume:

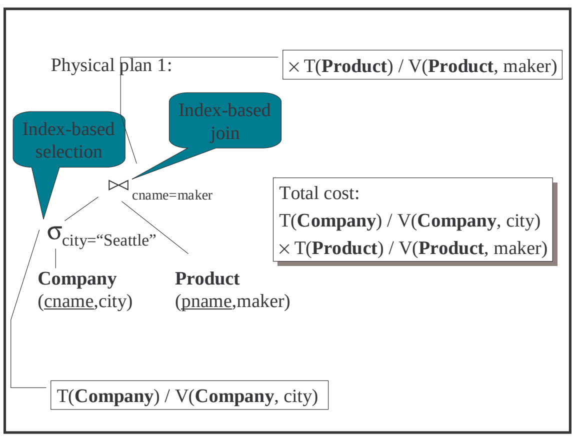
Clustered index: **Product**.<u>pname</u>, **Company**.<u>cname</u>
Unclustered index: **Product**.maker, **Company**.city

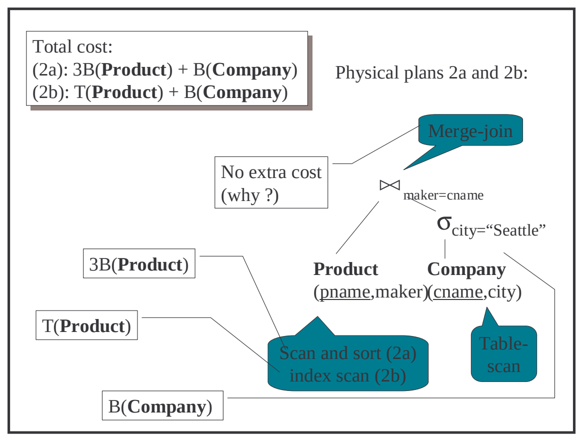
Logical Plan:











Plan 1: T(**Company**)/V(**Company**,city) × T(**Product**)/V(**Product**,maker)
Plan 2a: B(**Company**) + 3B(**Product**)
Plan 2b: B(**Company**) + T(**Product**)

Which one is better ??

It depends on the data !!

Example

T(**Company**) = 5,000 B(**Company**) = 500 M = 100
T(**Product**) = 100,000 B(**Product**) = 1,000

We may assume V(**Product**, maker) ≈ T(**Company**) (why ?)

- Case 1: V(**Company**, city) ≈ T(**Company**)

V(**Company**,city) = 2,000

- Case 2: V(**Company**, city) << T(**Company**)

V(**Company**,city) = 20

Which Plan is Best ?

Plan 1: T(**Company**)/V(**Company**,city) × T(**Product**)/V(**Product**,maker)
Plan 2a: B(**Company**) + 3B(**Product**)
Plan 2b: B(**Company**) + T(**Product**)

Case 1: Plan 1 = 2.5 * 20 = 50
Plan 2 = 500 + 3000 = 3500
Plan 3 = 500 + 100000 = 100500

Case 2: Plan 1 = 250 * 20 = 5000

Lessons

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic "best" plan: depends on the data
- In order to make the right choice
  - need to have ***statistics*** over the data
  - the B's, the T's, the V's