Lecture 26:

Wednesday, December 4, 2002

Administrative

- As per email to class:
  - Question 1b on the homework has been updated, see website

Outline

- Cost-based Query Optimziation
- Completing the physical query plan: 16.7
- Cost estimation: 16.4

Reducing the Search Space

- Left-linear trees v.s. Bushy trees
- Trees without cartesian product

Example: R(A,B) ⋈ S(B,C) ⋈ T(C,D)

Plan: (R(A,B) ⋈ T(C,D)) ⋈ S(B,C) has a cartesian product – most query optimizers will not consider it

Counting the Number of Join Orders

- The mathematics we need to know:

Given x_1, x_2, ..., x_n, how many permutations can we construct ? Answer: n!

- Example: permutations of $x_1x_2x_3x_4$ are:
  - $x_1x_2x_3x_4$
  - $x_2x_4x_3x_1$
  - . . .
  - (there are 4! = 4*3*2*1 = 24 permutations)

Counting the Number of Join Orders

- The mathematics we need to know:

Given the product $x_0x_1x_2...x_n$, in how many ways can we place n pairs of parenthesis around them ? Answer: $1/(n+1)*C^{2n}_n = (2n)!/((n+1)*(n!)^2)$

- Example: for n=3
  - $((x_0x_1)(x_2x_3))$
  - $((x_0(x_1x_2))x_3)$
  - $(x_0((x_1x_2)x_3))$
  - $((x_0x_1)x_2)x_3)$
  - $(x_0(x_1(x_2x_3)))$
- There are 6!/(4*3!*3!) = 5 ways

Counting the Number of Join Orders (Excercise)

$R_0(A_0,A_1) \bowtie R_1(A_1,A_2) \bowtie \ldots \bowtie R_n(A_n,A_{n+1})$

- The number of left linear join trees is:
- The number of left linear join trees without cartesian products is:
- The number of bushy join trees is:
- The number of bushy join trees without cartesian product is:

7

Number of Subplans Inspected by Dynamic Programming

$R_0(A_0,A_1) \bowtie R_1(A_1,A_2) \bowtie \ldots \bowtie R_n(A_n,A_{n+1})$

- The number of left linear subplans inspected is:
- The number of left linear subplans without cartesian products inspected is:
- The number of bushy join subplans inspected is:
- The number of bushy join subplans without cartesian product:

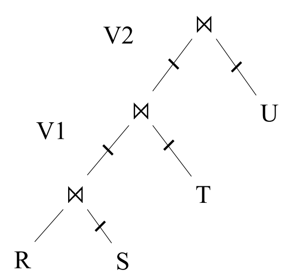
8

Completing the Physical Query Plan

- Choose algorithm to implement each operator
  - Need to account for more than cost:
    - How much memory do we have ?
    - Are the input operand(s) sorted ?
- Decide for each intermediate result:
  - To materialize
  - To pipeline

9

Materialize Intermediate Results Between Operators




```
HashTable ← S
repeat   read(R, x)
         y ← join(HashTable, x)
         write(V1, y)

HashTable ← T
repeat   read(V1, y)
         z ← join(HashTable, y)
         write(V2, z)

HashTable ← U
repeat   read(V2, z)
         u ← join(HashTable, z)
         write(Answer, u)
```

10

Materialize Intermediate Results Between Operators

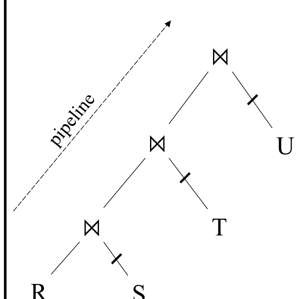
Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

11

Pipeline Between Operators




```
HashTable1 ← S
HashTable2 ← T
HashTable3 ← U
repeat   read(R, x)
         y ← join(HashTable1, x)
         z ← join(HashTable2, y)
         u ← join(HashTable3, z)
         write(Answer, u)
```

12

Pipeline Between Operators

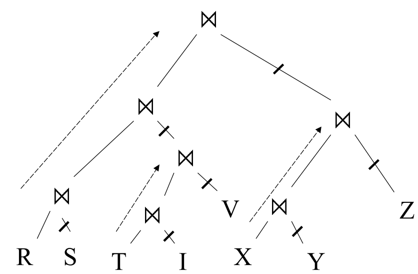
Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

13

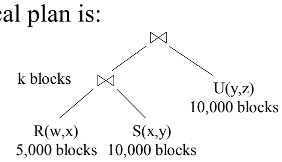
Pipeline in Bushy Trees



14

Example 16.36

- Logical plan is:

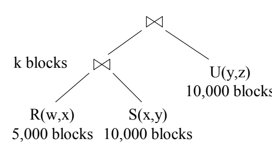


- Main memory M = 101 buffers

15

Example 16.36

M = 101



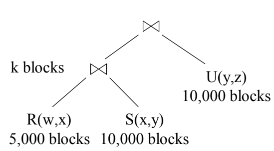
Naïve evaluation:

- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

16

Example 16.36

M = 101



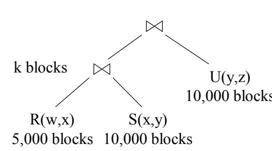
Smarter:

- Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
- Step 2: hash S on x into 100 buckets; to disk
- Step 3: read each R_i in memory (50 buffer) join with S_i (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we *pipeline*
- Cost so far: 3B(R) + 3B(S)

17

Example 16.36

M = 101



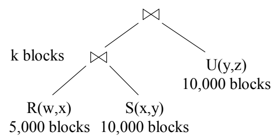
Continuing:

- How large are the 50 buckets on y ? Answer: k/50.
- If k <= 50 then keep all 50 buckets in Step 3 in memory, then:
- Step 4: read U from disk, hash on y and join with memory
- Total cost: 3B(R) + 3B(S) + B(U) = 55,000

18

Example 16.36

M = 101

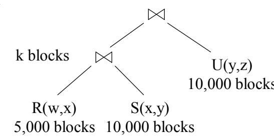


Continuing:
- If 50 < k <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size k/50 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: 3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k

19

Example 16.36

M = 101



Continuing:
- If k > 5000 then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

20

Example 16.36

Summary:

• If k <= 50,	cost = 55,000
• If 50 < k <=5000,	cost = 75,000 + 2k
• If k > 5000,	cost = 75,000 + 4k

21

Estimating Sizes

- Need size in order to estimate cost
- Example:
  - Cost of partitioned hash-join E1 ⋈ E2 is 3B(E1) + 3B(E2)
  - B(E1) = T(E1) * record size/ block size
  - B(E2) = T(E2) * record size/ block size
  - So, we need to estimate T(E1), T(E2)

22

Estimating Sizes

Estimating the size of a projection
- Easy: $T(\Pi_L(R)) = T(R)$
- This is because a projection doesn't eliminate duplicates

23

Estimating Sizes

Estimating the size of a selection
- $S = \sigma_{A=c}(R)$
  - T(S) san be anything from 0 to T(R) – V(R,A) + 1
  - Mean value: T(S) = T(R)/V(R,A)
- $S = \sigma_{A<c}(R)$
  - T(S) can be anything from 0 to T(R)
  - Heuristics: T(S) = T(R)/3

24

Estimating Sizes

Estimating the size of a natural join, $R \bowtie_A S$

- When the set of A values are disjoint, then $T(R \bowtie_A S) = 0$
- When A is a key in S and a foreign key in R, then $T(R \bowtie_A S) = T(R)$
- When A has a unique value, the same in R and S, then $T(R \bowtie_A S) = T(R)\ T(S)$

25

Estimating Sizes

Assumptions:

- *Containment of values*: if V(R,A) <= V(S,A), then the set of A values of R is included in the set of A values of S
  - Note: this indeed holds when A is a foreign key in R, and a key in S
- *Preservation of values*: for any other attribute B, $V(R \bowtie_A S, B) = V(R, B)$ (or V(S, B))

26

Estimating Sizes

Assume V(R,A) <= V(S,A)

- Then each tuple t in R joins *some* tuple(s) in S
  - How many ?
  - On average T(S)/V(S,A)
  - t will contribute T(S)/V(S,A) tuples in $R \bowtie_A S$
- Hence $T(R \bowtie_A S) = T(R)\ T(S) / V(S,A)$

In general: $T(R \bowtie_A S) = T(R)\ T(S) / \max(V(R,A),V(S,A))$

27

Estimating Sizes

Example:

- T(R) = 10000, T(S) = 20000
- V(R,A) = 100, V(S,A) = 200
- How large is $R \bowtie_A S$?

Answer: $T(R \bowtie_A S) = 10000\ 20000/200 = 1M$

28

Estimating Sizes

Joins on more than one attribute:

- $T(R \bowtie_{A,B} S) =$

  $T(R)\ T(S)/(\max(V(R,A),V(S,A))*\max(V(R,B),V(S,B)))$

29

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

30

Histograms

Employee(<u>ssn</u>, name, salary, phone)

- Maintain a histogram on salary:

Salary:	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
Tuples	200	800	5000	12000	6500	500

- T(Employee) = 25000, but now we know the distribution

31

Histograms

Ranks(rankName, salary)

- Estimate the size of Employee $\bowtie_{Salary}$ Ranks

Employee	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	200	800	5000	12000	6500	500

Ranks	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	8	20	40	80	100	2

32

Histograms

- Assume:
  - V(Employee, Salary) = 200
  - V(Ranks, Salary) = 250
- Main property:

Employee $\bowtie_{Salary}$ Ranks=$Employee_1 \bowtie_{Salary} Ranks_1$' $\cup$ ... $\cup$ $Emplyee_6 \bowtie_{Salary} Ranks_6$'

- A tuple t in $Employee_1$ joins with how many tuples in $Ranks_1$' ?
  - Answer: with $T(Employee_1)$/T(Employee) * T(Employee)/250 = T_1 / 250
- Then T(Employee $\bowtie_{Salary}$ Ranks) =

  $= \Sigma_{i=1,6} T_i T_i' / 250$

  = (200x8 + 800x20 + 5000x40 +
  12000x80 + 6500x100 + 500x2)/250

  = ….

33