Lecture 25:

Monday, December 27, 2002

1

Administrative

- Homework 5 is due on Monday, 12/9
- Project demos will be on Tuesday 12/10
  - 10am – 12pm: 6teams
  - 2pm – 4pm: 6teams
  - 4pm – 6pm: 6teams
- Please send me email for an appointment
  - First come first served…

2

Outline

- Cost-based optimization: 16.5, 16.6
- Completing the physical query plan: 16.7
- Cost estimation: 16.4 (will continue next time)

3

Cost-based Optimizations

- Main idea: apply algebraic laws, until estimated cost is minimal
- Practically: start from partial plans, introduce operators one by one
  - Will see in a few slides
- Problem: there are too many ways to apply the laws, hence too many (partial) plans

4

Cost-based Optimizations

Approaches:

- **Top-down**: the partial plan is a top fragment of the logical plan
- **Bottom up**: the partial plan is a bottom fragment of the logical plan

5

Search Strategies

- **Branch-and-bound**:
  - Remember the cheapest complete plan P seen so far and its cost C
  - Stop generating partial plans whose cost is > C
  - If a cheaper complete plan is found, replace P, C
- **Hill climbing**:
  - Remember only the cheapest partial plan seen so far
- **Dynamic programming**:
  - Remember the all cheapest partial plans

6

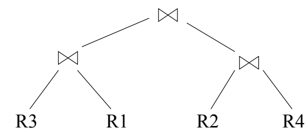
Dynamic Programming

Unit of Optimization: select-project-join

- Push selections down, pull projections up

Join Trees

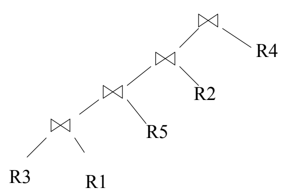
- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:



- A plan = a join tree
- A partial plan = a subtree of a join tree

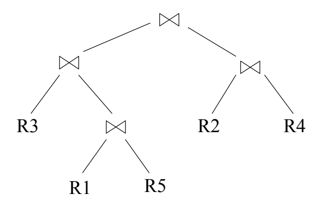
Types of Join Trees

- Left deep:



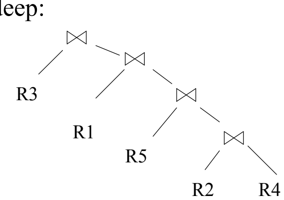
Types of Join Trees

- Bushy:



Types of Join Trees

- Right deep:



Problem

- Given: a query R1 ⋈ R2 ⋈ … ⋈ Rn
- Assume we have a function cost() that gives us the cost of every join tree
- Find the best join tree for the query

Dynamic Programming

- Idea: for each subset of {R1, …, Rn}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for {R1}, {R2}, …, {Rn}
  - Step 2: for {R1,R2}, {R1,R3}, …, {Rn-1, Rn}
  - …
  - Step n: for {R1, …, Rn}
- It is a bottom-up strategy
- A subset of {R1, …, Rn} is also called a *subquery*

13

Dynamic Programming

- For each subquery Q ⊆ {R1, …, Rn} compute the following:
  - Size(Q)
  - A best plan for Q: Plan(Q)
  - The cost of that plan: Cost(Q)

14

Dynamic Programming

- **Step 1**: For each {Ri} do:
  - Size({Ri}) = B(Ri)
  - Plan({Ri}) = Ri
  - Cost({Ri}) = (cost of scanning Ri)

15

Dynamic Programming

- **Step i**: For each Q ⊆ {R1, …, Rn} of cardinality i do:
  - Compute Size(Q) (later…)
  - For every pair of subqueries Q', Q'' s.t. Q = Q' ∪ Q'' compute cost(Plan(Q') ⋈ Plan(Q''))
  - Cost(Q) = the smallest such cost
  - Plan(Q) = the corresponding plan

16

Dynamic Programming

- Return Plan({R1, …, Rn})

17

Dynamic Programming

To illustrate, we will make the following simplifications:

- Cost(P1 ⋈ P2) = Cost(P1) + Cost(P2) + size(intermediate result(s))
- Intermediate results:
  - If P1 = a join, then the size of the intermediate result is size(P1), otherwise the size is 0
  - Similarly for P2
- Cost of a scan = 0

18

Dynamic Programming

- Example:
- Cost(R5 ⋈ R7) = 0 (no intermediate results)
- Cost((R2 ⋈ R1) ⋈ R7)
  = Cost(R2 ⋈ R1) + Cost(R7) + size(R2 ⋈ R1)
  = size(R2 ⋈ R1)

19

Dynamic Programming

- Relations: R, S, T, U
- Number of tuples: 2000, 5000, 3000, 1000
- Size estimation: T(A ⋈ B) = 0.01*T(A)*T(B)

20

Subquery	Size	Cost	Plan
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

21

Subquery	Size	Cost	Plan
RS	100k	0	RS
RT	60k	0	RT
RU	20k	0	RU
ST	150k	0	ST
SU	50k	0	SU
TU	30k	0	TU
RST	3M	60k	(RT)S
RSU	1M	20k	(RU)S
RTU	0.6M	20k	(RU)T
STU	1.5M	30k	(TU)S
RSTU	30M	60k+50k=110k	(RT)(SU)

22

Dynamic Programming

- Summary: computes optimal plans for subqueries:
  - Step 1: {R1}, {R2}, …, {Rn}
  - Step 2: {R1, R2}, {R1, R3}, …, {Rn-1, Rn}
  - …
  - Step n: {R1, …, Rn}
- We used naïve size/cost estimations
- In practice:
  - more realistic size/cost estimations (next time)
  - heuristics for Reducing the Search Space
    - Restrict to left linear trees
    - Restrict to trees "without cartesian product"
  - need more than just one plan for each subquery:
    - "interesting orders"

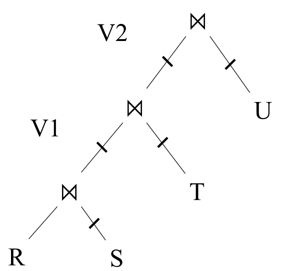
23

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

24

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)
```

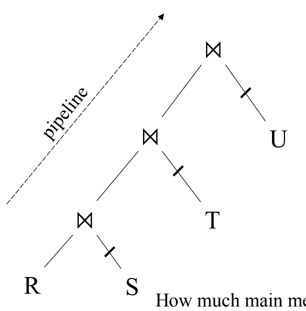

Materialize Intermediate Results Between Operators

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 =

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)
```

How much main memory do we need ? M =

Pipeline Between Operators

Question in class

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

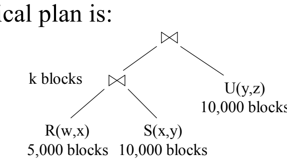
- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

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

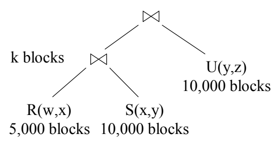
Example 16.36

- Logical plan is:



- Main memory M = 101 buffers

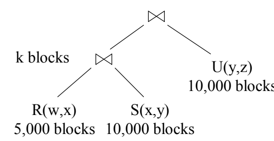
Example 16.36



Naïve evaluation:

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

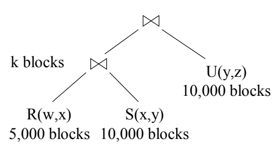
Example 16.36



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 Ri in memory (50 buffer) join with Si (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we *pipeline*
- Cost so far: 3B(R) + 3B(S)

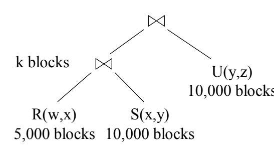
Example 16.36



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

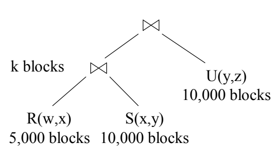
Example 16.36



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

Example 16.36



Continuing:

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

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

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)/ block size
  - B(E2) = T(E2)/ block size
  - So, we need to estimate T(E1), T(E2)

37

Estimating Sizes

Estimating the size of a projection

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

38

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

39

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 $T(R \bowtie_A S) = T(R)\ T(S)$

40

Estimating Sizes

Assumptions:

- <u>*Containment of values*</u>: 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
- <u>*Preservation of values*</u>: for any other attribute B, $V(R \bowtie_A S, B) = V(R, B)$ (or V(S, B))

41

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 S/V(S,A)
  - t will contribute 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: $T(R \bowtie_A S) = T(R)\ T(S) / \max(V(R,A),V(S,A))$

42

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

43

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))

44

Histograms

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

45

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

46

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

47

Histograms

- Assume:
  - V(Employee, Salary) = 200
  - V(Ranks, Salary) = 250
- Then T(Employee $\bowtie_{Salary}$ Ranks) =
  = $\Sigma_{i=1,6}$ T_i T_i' / 250
  = (200x8 + 800x20 + 5000x40 + 12000x80 + 6500x100 + 500x2)/250
  = ….

48