CIS 560 – Database System Concepts

Lecture 28

Query Optimization

November 8, 2013

Credits for slides: Chang, Ullman, Whitehead.

Planning

- Assignment 8 (indexes) due 11/8
- Project - DB design revision due 11/11
  - No class that day – use the time to work on project
- Assignment 9 (query optimization) due 11/15
- Exam 2 (assignments 6-9) – 11/20
- Project - DB implementation and queries due 11/22
- Quiz from special topics – 12/06
- Project presentations 12/9, 12/11, 12/13
- Project reports – finals week

2

Summary of Join Algorithms

- Nested Loop Join: B(R)+B(R)B(S)/M
  - Assuming block-at-a-time refinement, with one block-at-a time, the cost is: B(R)+ B(R)B(S)
- Hash Join: 3B(R) + 3B(S)
  - Assuming: min(B(R), B(S)) $\leq$ M^2
- Sort-Merge Join: 3B(R) + 3B(S)
  - Assuming B(R)+B(S) $\leq$ M^2
- Index Nested Loop Join: B(R) + T(R)B(S)/V(S,a)
  - Assuming S has clustered index on attribute a

Query Optimization Goal

- For a query
  - There exists many logical and physical query plans
  - Query optimizer needs to pick a good one

Example

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Manhattan'
   and x.sstate = 'KS'
```

- Some statistics
  - T(Supplier) = 1000 records
  - T(Supply) = 10,000 records
  - B(Supplier) = 100 pages
  - B(Supply) = 100 pages
  - V(Supplier,scity) = 20, V(Supplier,state) = 10
  - V(Supply,pno) = 2,500
  - Both relations are clustered
- M = 10

Relational Algebra Expressions

$$\pi_{sname}(\sigma_{scity=\text{'Manhattan'} \wedge \ sstate=\text{'KS'} \wedge \ pno=2} (\text{Supplier} \bowtie_{sid = sid} \text{Supply}))$$

$$\pi_{\ sname}((\sigma_{scity=\text{'Manhattan'} \wedge sstate=\text{'KS'}}(\text{Supplier})) \bowtie_{sid = sid} (\sigma_{\ pno=2} (\text{Supply})))$$

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 1

(On the fly) π_{sname}

Selection and project on-the-fly
-> No additional cost.

(On the fly)

$\sigma_{\text{scity='Manhattan'} \wedge \text{sstate='KS'} \wedge \text{pno=2}}$

(Block-nested loop) $\bowtie_{\text{sid = sid}}$

Total cost of plan is thus cost of join:
= B(Supplier)+B(Supplier)*B(Supply)/M
= 100 + 100 * 100 / 10
= 1,100 I/Os

Supplier
(File scan)

Supply
(File scan)

T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 2

(On the fly) π_{sname} (4)

Total cost
= 100 + 100 * 1/20 * 1/10 (1)
+ 100 + 100 * 1/2500 (2)
+ 2 (3)
+ 0 (4)
Total cost ≈ **204 I/Os**

(Sort-merge join) $\bowtie_{\text{sid = sid}}$ (3)

(Scan
write to T1)

(1) $\sigma_{\text{scity='Manhattan'} \wedge \text{sstate='KS'}}$

(Scan
write to T2)

(2) $\sigma_{\text{pno=2}}$

Supplier
(File scan)

Supply
(File scan)

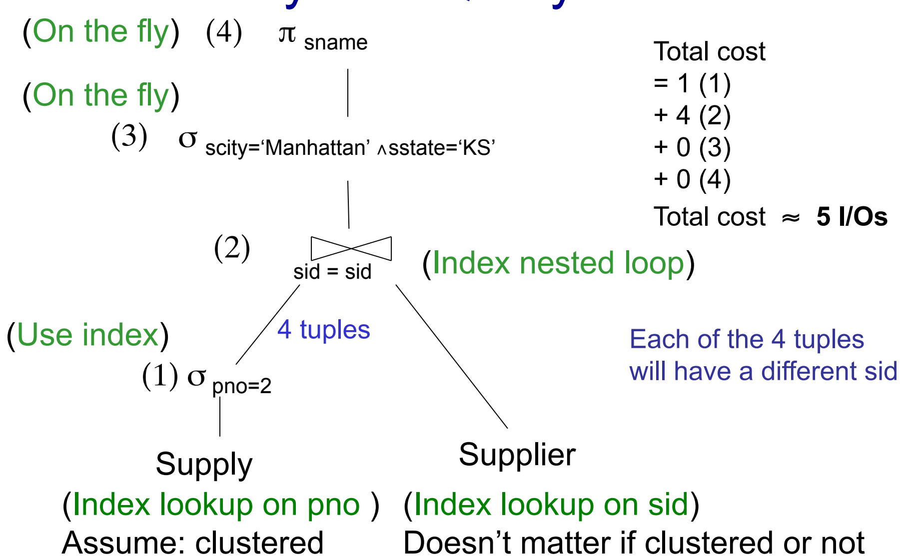
T(Supplier) = 1000
T(Supply) = 10,000

B(Supplier) = 100
B(Supply) = 100

V(Supplier,scity) = 20
V(Supplier,state) = 10
V(Supply,pno) = 2,500

M = 10

Physical Query Plan 3



Simplifications

- In the previous examples, we assumed that all index pages were in memory
- When this is not the case, we need to add the cost of fetching index pages from disk

Lessons

- Need to consider several physical plans
  - 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

Query Optimization Algorithm

- Enumerate alternative plans
- Compute estimated cost of each plan
  - Compute number of I/Os
- Choose plan with lowest cost
  - This is called cost-based optimization

Components of an optimizer

We need three things in an optimizer:

- Search space (algebraic laws – relational algebra equivalences)
- Algorithm for enumerating query plans
- A cost estimator for a plan

Relational Algebra Equivalences

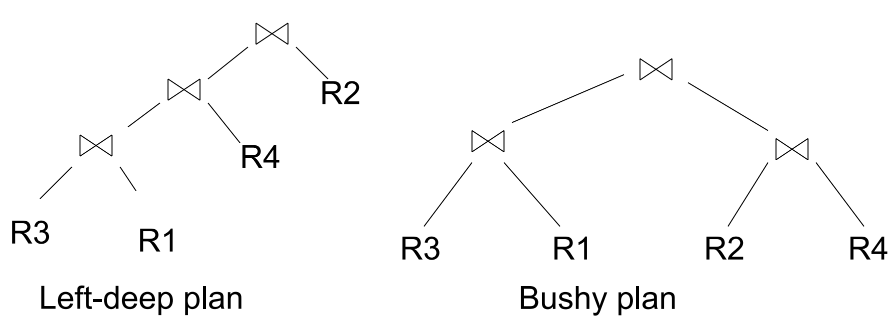
- We can commute and combine operators
- We just have to be careful that the fields we need are available when we apply the operator

Commutativity, Associativity, Distributivity

$R \cup S = S \cup R$, $R \cup (S \cup T) = (R \cup S) \cup T$
$R \cap S = S \cap R$, $R \cap (S \cap T) = (R \cap S) \cap T$
$R \bowtie S = S \bowtie R$, $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Left-Deep Plans and Bushy Plans



Example

Which plan is more efficient?
$R \bowtie (S \bowtie T)$ or $(R \bowtie S) \bowtie T$?

- Assumptions:
  - Every join selectivity is 10%
    - That is: $T(R \bowtie S) = 0.1 * T(R) * T(S)$ etc.
  - $B(R)=100$, $B(S) = 50$, $B(T)=500$
  - All joins are main memory joins
  - All intermediate results are materialized

Laws involving selection:

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$$
$$\sigma_{C \text{ AND } C'}(R) = \sigma_{C}(\sigma_{C'}(R)) = \sigma_{C}(R) \cap \sigma_{C'}(R)$$
$$\sigma_{C \text{ OR } C'}(R) = \sigma_{C}(R) \cup \sigma_{C'}(R)$$
$$\sigma_{C}(R \cup S) = \sigma_{C}(R) \cup \sigma_{C}(S)$$

When C involves only attributes of R

$$\sigma_{C}(R \cup S) = \sigma_{C}(R) \cup S$$
$$\sigma_{C}(R - S) = \sigma_{C}(R) - S$$
$$\sigma_{C}(R \bowtie S) = \sigma_{C}(R) \bowtie S$$

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

  $\sigma_{F=3}(R \bowtie_{D=E} S) =$?

  $\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) =$?

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

  $\sigma_{F=3}(R \bowtie_{D=E} S) = R \bowtie_{D=E} (\sigma_{F=3}(S))$

  $\sigma_{A=5 \text{ AND } G=9}(R \bowtie_{D=E} S) = \sigma_{A=5}(\sigma_{G=9}(R \bowtie_{D=E} S))$

  $= (\sigma_{A=5}(R)) \bowtie_{D=E} (\sigma_{G=9}(S))$

Laws Involving Projections

$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$
$\Pi_M(\Pi_N(R)) = \Pi_M(R)$ /* note that M ⊆ N */

- Example R(A,B,C,D), S(E, F, G)

  $\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{?}(\Pi_{?}(R) \bowtie_{D=E} \Pi_{?}(S))$

Laws Involving Projections

$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$
$\Pi_M(\Pi_N(R)) = \Pi_M(R)$ /* note that M ⊆ N */

- Example R(A,B,C,D), S(E, F, G)

  $\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{A,B,G}(\Pi_{A,B,D}(R) \bowtie_{D=E} \Pi_{E,G}(S))$

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition
- Cannot consider ALL plans
  - Heuristics: only partial plans with “low” cost

Algorithms for enumerating plans: key decisions

- Logical plan
  - What logical plans do we consider (left-deep, bushy?)
    - *Search space*
  - Which algebraic laws do we apply, and in which context(s)?
    - *Optimization rules*
  - In what order do we explore the search space?
    - *Optimization algorithm*
- Physical plan
  - What join algorithms to use?
  - What access paths to use (file scan or index)?

Types of Optimizers

- Rule-based optimizers:
  - Apply greedily rules that always improve
    - Typically: push selections down, pull projections up
  - Very limited: no longer used today
- Cost-based optimizers
  - Use a cost model to estimate the cost of each plan
  - Select the “cheapest” plan

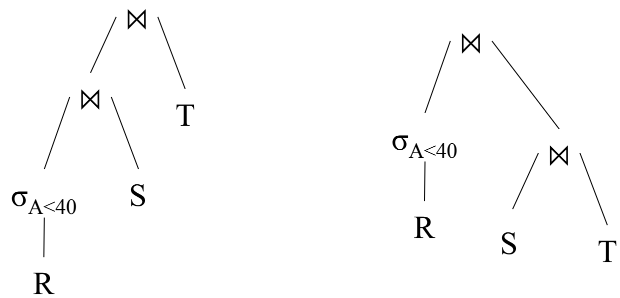
The Search Space

- Complete plans
- Bottom-up plans
- Top-down plans

Complete Plans

R(A,B)
S(B,C)
T(C,D)

```
SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40
```



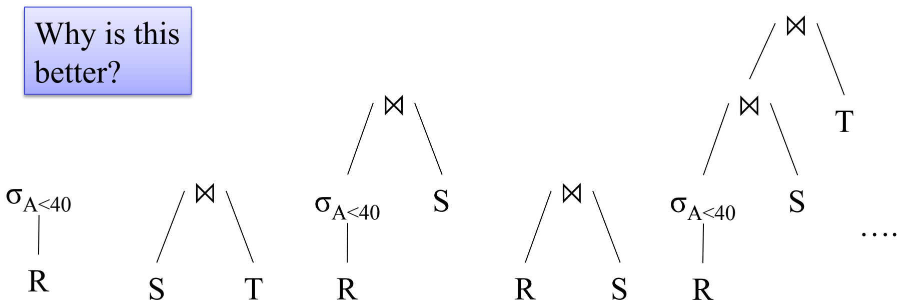
Why is this search space inefficient?

Bottom-up Partial Plans

R(A,B)
S(B,C)
T(C,D)

```
SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40
```

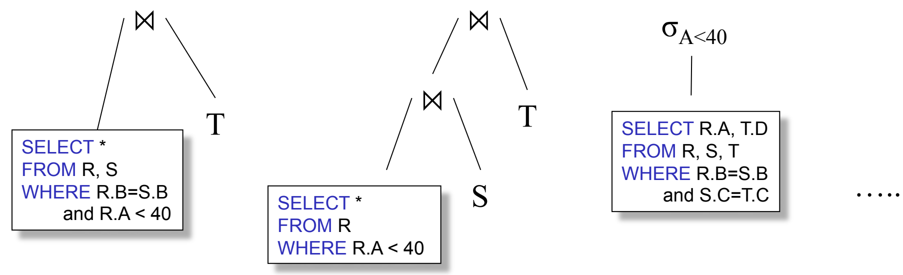
Why is this better?



Top-down Partial Plans

R(A,B)
S(B,C)
T(C,D)

```
SELECT *
FROM R, S, T
WHERE R.B=S.B and S.C=T.C and R.A<40
```



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 all cheapest partial plans

Dynamic Programming

Originally proposed in System R [1979]

- Limited to joins: *join reordering algorithm*
- Bottom-up
- Only handles single block queries:

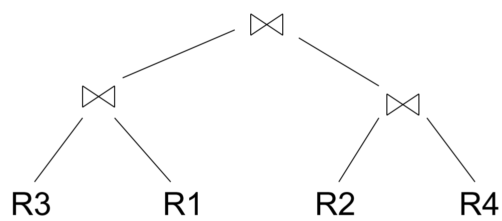
```
SELECT list
FROM   R1, …, Rn
WHERE cond1 AND cond2 AND . . . AND condk
```

Dynamic Programming

- Search space = join trees
- Algebraic laws = commutativity, associativity
- Algorithm = dynamic programming ☺

Join Trees

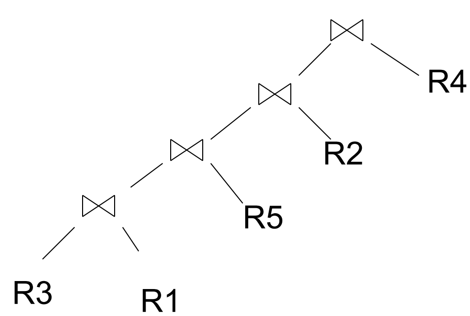
- R1 ⋈ R2 ⋈ …. ⋈ Rn
- Join tree:



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

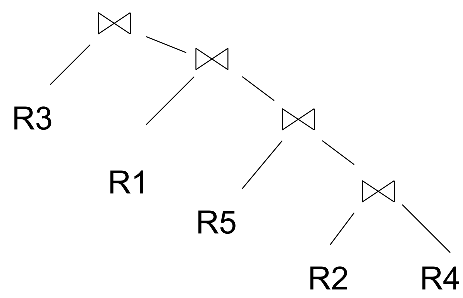
Types of Join Trees

- Left deep:



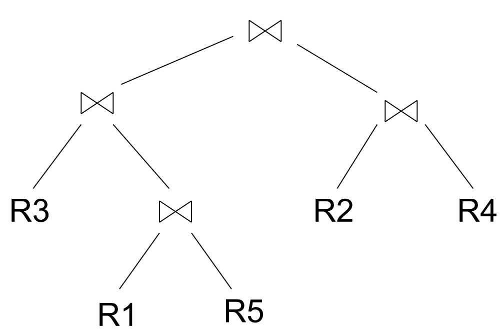
Types of Join Trees

- Right deep:



Types of Join Trees

- Bushy:



SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Dynamic Programming

Join ordering:

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

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

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*

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Dynamic Programming

- For each subquery Q $\subseteq${R1, …, Rn} compute the following:
  - Size(Q) = the estimated size of Q
  - Plan(Q) = a best plan for Q
  - Cost(Q) = the estimated cost of that plan

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Dynamic Programming

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

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Dynamic Programming

- **Step i**: For each Q ⊆{R_1, …, R_n} of cardinality i do:
  - Size(Q) = estimate it recursively
  - 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

SELECT list
FROM R1, …, Rn
WHERE $cond_1$ AND $cond_2$ AND . . . AND $cond_k$

Dynamic Programming

- **After step n**: Return Plan({R_1, …, R_n})

Example

To illustrate, ad-hoc cost model (from the book ☺):

- In practice: more realistic size/cost estimations

- $\text{Cost}(P_1 \bowtie P_2) = \text{Cost}(P_1) + \text{Cost}(P_2) +$
  $\text{size(intermediate results for } P_1, P_2)$
  - Intermediate results:
    - If P1 is 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

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)

SELECT *
FROM R, S, T, U
WHERE $cond_1$ AND $cond_2$ AND . . .

Example

- R ⋈ S ⋈ T ⋈ U
- Assumptions:

T(R) = 2000
T(S) = 5000
T(T) = 3000
T(U) = 1000

All join selectivities = 1%

T(R ⋈ S) = 0.01*T(R)*T(S)
T(S ⋈ T) = 0.01*T(S)*T(T)
etc.

T(R) = 2000
T(S) = 5000
T(T) = 3000
T(U) = 1000

T(R ⋈ S) = 0.01*T(R)*T(S)
T(S ⋈ T) = 0.01*T(S)*T(T)
etc.

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

T(R) = 2000
T(S) = 5000
T(T) = 3000
T(U) = 1000

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)

$T(R \bowtie S) = 0.01*T(R)*T(S)$
$T(S \bowtie T) = 0.01*T(S)*T(T)$
etc.

Reducing the Search Space

- Restriction 1: only left linear trees (no bushy)
- Restriction 2: no trees with cartesian product

$R(A,B) \bowtie S(B,C) \bowtie T(C,D)$

Plan: $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$
has a cartesian product.
Most query optimizers will not consider it

Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)
- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further

Completing the Physical Query Plan

- Choose algorithm for each operator
  - How much memory do we have?
  - Are the input operand(s) sorted?
- Access path selection for base tables
- Decide for each intermediate result:
  - To materialize
  - To pipeline

Summary of Query Optimization

- Three parts:
  - search space, algorithms, size/cost estimation
- Ideal goal: find optimal plan. But
  - Impossible to estimate accurately
  - Impossible to search the entire space
- Goal of today's optimizers:
  - Avoid very bad plans