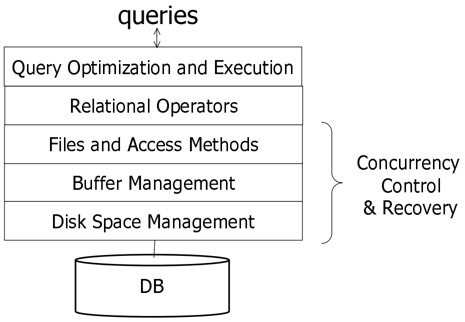
CS 245: Database System Principles

Notes 7: Query Optimization

Chris Olston
Hector Garcia-Molina

System Structure



queries

Query Optimization and Execution

Relational Operators

Files and Access Methods

Buffer Management

Disk Space Management

DB

Concurrency
Control
& Recovery

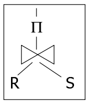
Metamorphosis: From Query to Plan

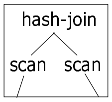
- Declarative query
  - SQL
- Logical query plan
  - Relational algebra
- Physical query plan
  - What actually gets executed

1:n

1:n

SELECT
FROM
WHERE





Query Interface

History

- In the "dark days," the application programmer would program the execution plan
- Problems:
  - More work for application programmer
  - Have to re-program to handle changes to physical design (e.g., remove an index)
  - Optimal execution plan may change over time

Declarative Query Languages

Now

- Relational databases have query language interface
- Declarative specification:
  - Say what you want, not how to get it

⇨ Rely on the query optimizer to pick the best plan …

Query Optimization Strategies

1) Guess
   - ➢ Could be orders of magnitude slower than the optimal plan (hours instead of seconds)
2) Ask user's advice
   - ➢ User does not know about physical design of DB
3) Execute all options, measure, and pick fastest
   - ➢ Optimal, but obviously not the way to go
4) Be smart!

Relational Query Optimization

Enumerate plans;
Estimate costs;
Pick cheapest

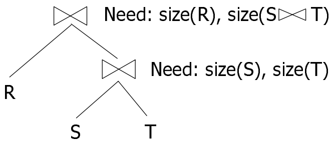
- Relies on cost estimation
- Costs estimated using:
  - Statistics about relations in DB
  - Input size estimates

Organization of Lecture

- Estimating costs
  - Estimating input sizes ← next
- Generating & comparing plans
  - Logical transformations & heuristics
  - Generating plans piece-wise
  - Pruning

Motivation: Input size estimation

- Cost of an operation (join, sort, …) depends on sizes of inputs



Estimating input size

- Keep statistics for relation R
  - T(R) = |R| = # tuples in R
  - S(R) = # of bytes in each R tuple
  - B(R) = # of blocks to hold all R tuples
  - V(R, A) = # distinct values in R for attribute A

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

A: 20 byte string
B: 4 byte integer
C: 8 byte date
D: 5 byte string

T(R) = 5 S(R) = 37
V(R,A) = 3 V(R,C) = 5
V(R,B) = 1 V(R,D) = 4

Size estimates for W = R1 x R2

T(W) = T(R1) × T(R2)

S(W) = S(R1) + S(R2)

Size estimate for $W = \sigma_{A=a}(R)$

$S(W) = S(R)$

$T(W) = ?$

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

V(R,A)=3
V(R,B)=1
V(R,C)=5
V(R,D)=4

$W = \sigma_{z=val}(R) \quad T(W) = \frac{T(R)}{V(R,Z)}$

Assumption:

Values in select expression Z = val are uniformly distributed over possible V(R,Z) values.

Alternate Assumption:

Values in select expression Z = val are uniformly distributed over domain with DOM(R,Z) values.

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

Alternate assumption
V(R,A)=3 DOM(R,A)=10
V(R,B)=1 DOM(R,B)=10
V(R,C)=5 DOM(R,C)=10
V(R,D)=4 DOM(R,D)=10

$W = \sigma_{z=val}(R) \quad T(W) = ?$

$C=val \Rightarrow T(W) = (1/10)1 + (1/10)1 + \ldots$
$= (5/10) = 0.5$

$B=val \Rightarrow T(W)= (1/10)5 + 0 + 0 = 0.5$

$A=val \Rightarrow T(W)= (1/10)2 + (1/10)2 + (1/10)1$
$= 0.5$

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

Alternate assumption
V(R,A)=3 DOM(R,A)=10
V(R,B)=1 DOM(R,B)=10
V(R,C)=5 DOM(R,C)=10
V(R,D)=4 DOM(R,D)=10

$$W = \sigma_{z=val}(R) \quad T(W) = \frac{T(R)}{DOM(R,Z)}$$

Selection cardinality

SC(R,A) = average # records that satisfy equality condition on R.A

$$SC(R,A) = \begin{cases} \frac{T(R)}{V(R,A)} \\ \\ \frac{T(R)}{DOM(R,A)} \end{cases}$$

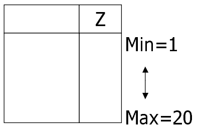
What about $W = \sigma_{z \geq val}(R)$?

T(W) = ?

- Solution # 1:
  T(W) = T(R)/2
- Solution # 2:
  T(W) = T(R)/3

- Solution # 3: Estimate values in range

Example R



V(R,Z)=10

$W = \sigma_{z \geq 15}(R)$

$$f = \frac{20-15+1}{20-1+1} = \frac{6}{20} \quad \text{(fraction of range)}$$

$T(W) = f \times T(R)$

Size estimate for $W = R1 \bowtie R2$

Let x = attributes of R1
y = attributes of R2

Case 1 $\quad X \cap Y = \varnothing$

Same as R1 x R2

Case 2 $\quad W = R1 \bowtie R2 \quad X \cap Y = A$

R1

A	B	C

R2

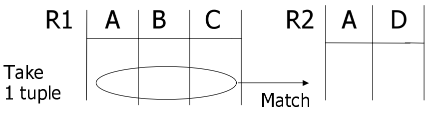
A	D

Assumption:

$V(R1,A) \leq V(R2,A) \Rightarrow$ Every A value in R1 is in R2
$V(R2,A) \leq V(R1,A) \Rightarrow$ Every A value in R2 is in R1

“containment of value sets” Sec. 7.4.4

Computing T(W) when $V(R1,A) \leq V(R2,A)$



1 tuple matches with $\frac{T(R2)}{V(R2,A)}$ tuples...

so $T(W) = \frac{T(R2)}{V(R2, A)} \times T(R1)$

- $V(R1,A) \leq V(R2,A)$ $T(W) = \frac{T(R2)\ T(R1)}{V(R2,A)}$
- $V(R2,A) \leq V(R1,A)$ $T(W) = \frac{T(R1)\ T(R2)}{V(R1,A)}$

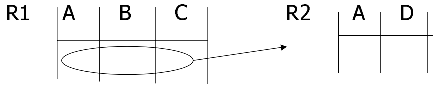
[A is common attribute]

In general $W = R1 \bowtie R2$

$$T(W) = \frac{T(R1)\ T(R2)}{\max\{\ V(R1,A),\ V(R2,A)\ \}}$$

$\boxed{\text{Case 2}}$ with alternate assumption

Values uniformly distributed over domain



This tuple matches T(R2)/DOM(R2,A) so

$$T(W) = \frac{T(R2)\ T(R1)}{DOM(R2,\ A)} = \frac{T(R2)\ T(R1)}{DOM(R1,\ A)}$$

Assume the same

In all cases:

$S(W) = S(R1) + S(R2) - S(A)$

size of attribute A

Using similar ideas, we can estimate sizes of:

$\Pi_{AB}(R)$ ..... Sec. 7.4.2

$\sigma_{A=a \wedge B=b}(R)$ .... Sec. 7.4.3

$R \bowtie S$ with common attribs. A,B,C
Sec. 7.4.5

Union, intersection, diff, …. Sec. 7.4.7

Note: for complex expressions, need intermediate T,S,V results.

E.g. $W = [\underbrace{\sigma_{A=a}(R1)}_{\text{Treat as relation U}}] \bowtie R2$

T(U) = T(R1)/V(R1,A)   S(U) = S(R1)

Also need V (U, *) !!

To estimate Vs

E.g., U = $\sigma_{A=a}$(R1)

Say R1 has attribs A,B,C,D

V(U, A) =
V(U, B) =
V(U, C) =
V(U, D) =

Example

R1

A	B	C	D
cat	1	10	10
cat	1	20	20
dog	1	30	10
dog	1	40	30
bat	1	50	10

V(R1,A)=3
V(R1,B)=1
V(R1,C)=5
V(R1,D)=3

U = $\sigma_{A=a}$(R1)

V(U,A) =1   V(U,B) =1   V(U,C) = $\frac{T(R1)}{V(R1,A)}$

V(D,U) ... somewhere in between

Possible Guess   U = $\sigma_{A=a}$(R)

V(U,A) = 1
V(U,B) = V(R,B)

For Joins   U = R1(A,B) $\bowtie$ R2(A,C)

V(U,A) = min { V(R1, A), V(R2, A) }
V(U,B) = V(R1, B)
V(U,C) = V(R2, C)

[called “preservation of value sets” in section 7.4.4]

Example:

Z = R1(A,B) $\bowtie$ R2(B,C) $\bowtie$ R3(C,D)

R1 T(R1) = 1000 V(R1,A)=50 V(R1,B)=100
R2 T(R2) = 2000 V(R2,B)=200 V(R2,C)=300
R3 T(R3) = 3000 V(R3,C)=90 V(R3,D)=500

Partial Result: $U = R \bowtie S$

$T(U) = \frac{1000\times2000}{200}$

V(U,A) = 50
V(U,B) = 100
V(U,C) = 300

$Z = U \bowtie R3$

$T(Z) = \frac{1000\times2000\times3000}{200\times300}$

V(Z,A) = 50
V(Z,B) = 100
V(Z,C) = 90
V(Z,D) = 500

Summary

- Estimating size of results is an “art”

- Don’t forget:
  Statistics must be kept up to date…
  (cost?)

Outline

- Estimating cost of query plan
  - Estimating size of inputs ← done!
  - Cost estimation ← next

Cost Estimation

Cost depends on input size/distribution and other parameters such as:

M = # memory blocks available
HT(i) = # levels in index i
LB(i) = # of leaf blocks in index i

Cost Formulas

- From Notes 6 …
  - Nested-Loops Join ($R \bowtie S$):

$$\text{Cost} = B(R) + \left\lceil \frac{B(R)}{M-2} \right\rceil \cdot B(S)$$

- Index-NL ($R \bowtie_{R.A = S.A} S$, index on S.A):

$$\text{Cost} = B(R) + |R| \cdot HT(\text{S-Index}) + \frac{|R| \cdot |S|}{\max\{V(R,A), V(S,A)\}}$$

(unless part of index buffered)

(unless S is clustered)

- Sort-Merge Join ($R \bowtie_{R.A = S.A} S$):

Cost to sort R into runs = 2·B(R)
Cost to sort S into runs = 2·B(S)
Cost to merge R & S runs = B(R) + B(S)

Overall Cost = 3·(B(R) + B(S))

- Hash Join ($R \bowtie_{R.A = S.A} S$):

Cost = 3·(B(R) + B(S))

(unless keep some buckets in memory:
hybrid hash join)

Outline

- Estimating cost of query plan ← done!
- Generating and comparing plans ← next

Generate & Compare Plans: One Idea

1. Enumerate all possible plans
2. Estimate costs
3. Pick best one

⇨ Problem: too many plans

Observation:
some plans are <u>almost</u> always bad…

Generate & Compare Plans: Better Idea

1. Selectively enumerate logical plans (using heuristics) → next
2. Enumerate physical alternatives
3. Estimate costs
4. Pick best one

Relational algebra transformations

- Transformation rules (preserve equivalence)
- What are good transformations?

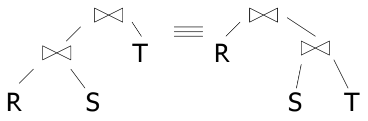
Rules: Natural joins & cross products & union

$R \bowtie S \quad = \quad S \bowtie R$

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

Note:

- Carry attribute names in results, so order is not important
- Can also write as trees, e.g.:



Rules: Natural joins & cross products & union

$R \bowtie S \quad = \quad S \bowtie R$

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

R x S = S x R

(R x S) x T = R x (S x T)

R U S = S U R

R U (S U T) = (R U S) U T

Rules: Selects

$\sigma_{p1 \wedge p2}(R) = \quad \sigma_{p1}\ [\ \sigma_{p2}(R)]$

$\sigma_{p1 \vee p2}(R) =$ See textbook

(gets complicated with bag semantics)

Rules: Project

Let: X = set of attributes
Y = set of attributes
XY = X U Y

$\pi_{xy}(R) = \xcancel{\pi_x [\pi_y (R)]}$

Rules: $\sigma + \bowtie$ combined

Let p = predicate with only R attribs
q = predicate with only S attribs
m = predicate with only R,S attribs

$\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$

$\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$

Rules: $\sigma + \bowtie$ combined (continued)

Some Rules can be Derived:

$\sigma_{p \wedge q} (R \bowtie S) =$

$\sigma_{p \wedge q \wedge m} (R \bowtie S) =$

$\sigma_{p \vee q} (R \bowtie S) =$

Do one, others left as exercises:

$\sigma_{p \wedge q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)]$

$\sigma_{p \wedge q \wedge m} (R \bowtie S) =$

$\sigma_m \left[(\sigma_p R) \bowtie (\sigma_q S)\right]$

$\sigma_{p \vee q} (R \bowtie S) =$

$\left[(\sigma_p R) \bowtie S\right] \cup \left[R \bowtie (\sigma_q S)\right]$

--> Derivation for first one:

$\sigma_{p \wedge q} (R \bowtie S) =$

$\sigma_p \left[\sigma_q (R \bowtie S)\right] =$

$\sigma_p \left[R \bowtie \sigma_q (S)\right] =$

$[\sigma_p (R)] \bowtie [\sigma_q (S)]$

Rules: π,σ combined

Let x = subset of R attributes
z = attributes in predicate P
(subset of R attributes)

$\pi_x[\sigma_{p}(R)] = \pi_x \left\{\sigma_p \left[\overset{\pi_{xz}}{\cancel{\pi_x}} (R)\right]\right\}$

Rules: $\pi, \bowtie$ combined

Let x = subset of R attributes
y = subset of S attributes
z = intersection of R,S attributes

$\pi_{xy} (R \bowtie S) =$

$\pi_{xy}\left\{[\pi_{xz}(R)] \bowtie [\pi_{yz}(S)]\right\}$

$\pi_{xy}\{\sigma_p\ (R \bowtie S)\}\ =$

$\pi_{xy}\{\sigma_p\ [\pi_{xz'}(R) \bowtie \pi_{yz'}(S)]\}$

$z' = z\ U\ \{\text{attributes used in P}\}$

Rules for σ, π combined with X

similar...

e.g., $\sigma_p\ (R\ X\ S) =\ ?$

Rules σ, U combined:

$\sigma_p(R\ U\ S) = \sigma_p(R)\ U\ \sigma_p(S)$

$\sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S)$

In textbook: more transformations

- Eliminate common sub-expressions
- Other operations: duplicate elimination

Which are "good" transformations?

☐ $\sigma_{p1 \wedge p2}\ (R) \rightarrow \sigma_{p1}\ [\sigma_{p2}\ (R)]$

☐ $\sigma_p\ (R \bowtie S) \rightarrow [\sigma_p\ (R)] \bowtie S$

☐ $R \bowtie S \rightarrow S \bowtie R$

☐ $\pi_x\ [\sigma_p\ (R)] \rightarrow \pi_x\ \{\sigma_p\ [\pi_{xz}\ (R)]\}$

Bottom line:

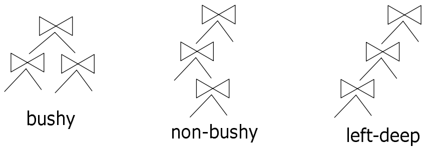
- There are some transformations that are usually good
- BUT: No transformation is always good
- Tradeoff:

reduced search space ⟺ loss of optimality

System R Optimizer Heuristics

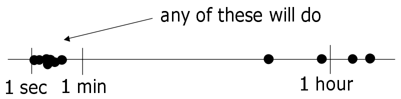
(usually good)

- Push down selections & projections
- Postpone cross-products
- Only consider "left-deep" query plans



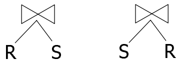
Query "Good-izer"

- Use of heuristics → can't claim optimality
- Instead, hope to get a fairly good plan
- Goal: eliminate horrible plans that take hours instead of seconds

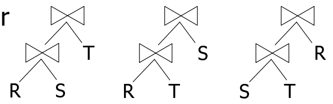


System R Optimizer: Transformations

- Left/right join inputs



- Join order



- Physical alternatives
  - Access methods (seq. scan, index lookup, …)
  - Join strategies (nested-loops, hash join, …)

Generate & Compare Plans

1. Selectively enumerate logical plans (using heuristics) → done
2. Enumerate physical alternatives → straightforward
3. Estimate costs
4. Pick best one

⇨ Problem: takes too long

Observation:
plans share pieces ("sub-plans") …

Generate & Compare Plans: Improved

Combine enumeration & cost estimation

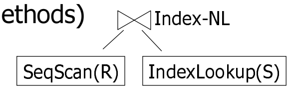
1. Enumerate small sub-plans & estimate costs
2. Prune (remove) "sub-optimal" alternatives
3. Enumerate ways to assemble sub-plans into larger sub-plans & estimate costs
4. Prune again (keep only "optimal" sub-plans)

… Keep building larger "optimal" sub-plans

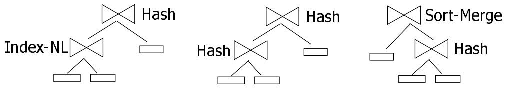
… Eventually generate "optimal" overall plan

Dynamic Programming Approach

- Level 1 sub-plan: join of 2 relations (plus access methods)



- Level 2 sub-plan: join of 3 relations

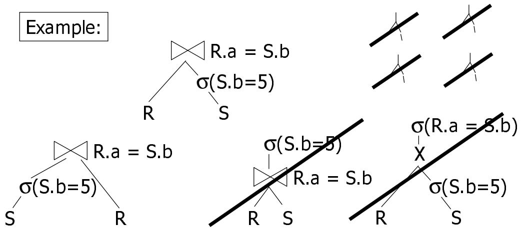


- Level n sub-plan: join of n+1 relations

Generating Logical Sub-Plans

- Only generate logical sub-plans that conform to heuristic rules

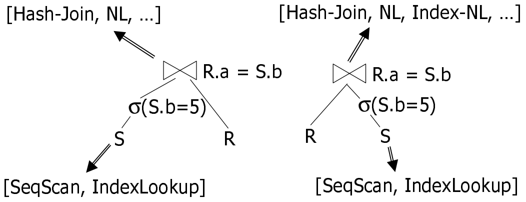
Example:



Generating Physical Sub-Plans

- For each heuristically chosen logical sub-plan, try all combinations of physical alternatives

Example:



Pruning

- Estimate cost of each physical sub-plan
- For each sub-plan with identical input and output, keep only:
  - Optimal plan overall
  - Optimal plan for each "interesting order"
    - Ordered on some field
- Discard the rest

Example

SELECT *
FROM R, S, T
WHERE R.A = S.A and S.B = T.B

- |R| = 30,000; B(R) = 300
- |S| = 100,000; B(S) = 1000
- |T| = 20,000; B(T) = 200
- V(S, B) = 25,000; V(T, B) = 10,000

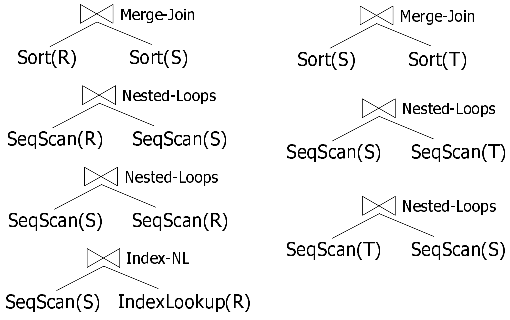
Example (cont.)

- Memory size: M = 102
- Index on R.A; all non-leaves fit in memory
- S.A foreign key onto R.A

For simplicity:

- Assume: join tuple size is sum of sizes of component tuples
- Assume: always write out intermediate results
- Consider the following join strategies:
  - Nested-loops
  - Index nested-loops
  - Sort-merge join

Level-1 Sub-Plans (no X-prods)



$\bowtie$ Merge-Join: Sort(R) Sort(S)

$\bowtie$ Nested-Loops: SeqScan(R) SeqScan(S)

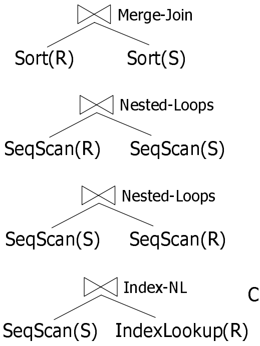
$\bowtie$ Nested-Loops: SeqScan(S) SeqScan(R)

$\bowtie$ Index-NL: SeqScan(S) IndexLookup(R)

$\bowtie$ Merge-Join: Sort(S) Sort(T)

$\bowtie$ Nested-Loops: SeqScan(S) SeqScan(T)

$\bowtie$ Nested-Loops: SeqScan(T) SeqScan(S)



$\bowtie$ Merge-Join: Sort(R) Sort(S)

Cost = 3·(B(R) + B(S)) = 3900

$\bowtie$ Nested-Loops: SeqScan(R) SeqScan(S)

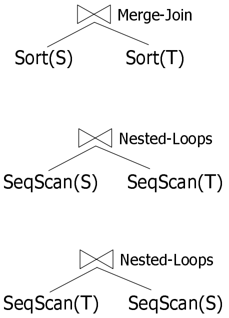
$$\text{Cost} = B(R) + \left\lceil \frac{B(R)}{M-2} \right\rceil \cdot B(S)$$
$$= 3300$$

$\bowtie$ Nested-Loops: SeqScan(S) SeqScan(R)

$$\text{Cost} = B(S) + \left\lceil \frac{B(S)}{M-2} \right\rceil \cdot B(R)$$
$$= 4000$$

$\bowtie$ Index-NL: SeqScan(S) IndexLookup(R)

Cost = B(S) + |S| + |S| = 201,000



$\bowtie$ Merge-Join: Sort(S) Sort(T)

Cost = 3·(B(S) + B(T)) = 3600

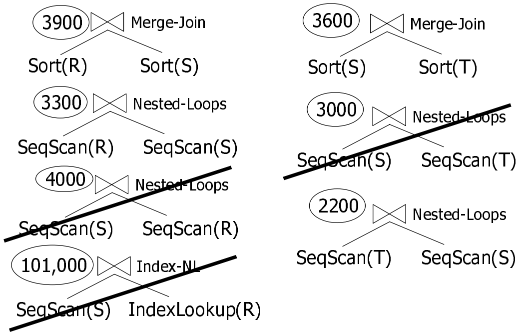
$\bowtie$ Nested-Loops: SeqScan(S) SeqScan(T)

$$\text{Cost} = B(S) + \left\lceil \frac{B(S)}{M-2} \right\rceil \cdot B(T)$$
$$= 3000$$

$\bowtie$ Nested-Loops: SeqScan(T) SeqScan(S)

$$\text{Cost} = B(T) + \left\lceil \frac{B(T)}{M-2} \right\rceil \cdot B(S)$$
$$= 2200$$

Pruning Level-1 Sub-Plans



3900 $\bowtie$ Merge-Join: Sort(R) Sort(S)

3300 $\bowtie$ Nested-Loops: SeqScan(R) SeqScan(S)

~~4000 $\bowtie$ Nested-Loops: SeqScan(S) SeqScan(R)~~

~~101,000 $\bowtie$ Index-NL: SeqScan(S) IndexLookup(R)~~

3600 $\bowtie$ Merge-Join: Sort(S) Sort(T)

~~3000 $\bowtie$ Nested-Loops: SeqScan(S) SeqScan(T)~~

2200 $\bowtie$ Nested-Loops: SeqScan(T) SeqScan(S)

What is size of R $\bowtie$ S?

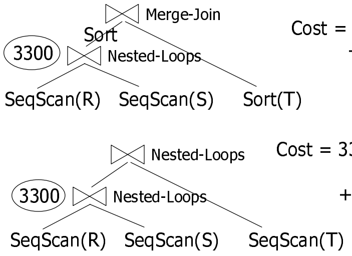
Recall: S.A foreign key onto R.A

$|R \bowtie S| = |S|$

B(RS) = 2·B(S)

Level-2 Sub-Plans (left-deep)

B(RS) = 2000



$\bowtie$ Merge-Join: Sort(3300 $\bowtie$ Nested-Loops: SeqScan(R) SeqScan(S)) Sort(T)

Cost = 3300 + B(RS)
+ 3·(B(RS) + B(T))
= 11,900

$\bowtie$ Nested-Loops: (3300 $\bowtie$ Nested-Loops: SeqScan(R) SeqScan(S)) SeqScan(T)

$$\text{Cost} = 3300 + B(RS) + B(RS) + \left\lceil \frac{B(RS)}{M-2} \right\rceil \cdot B(T)$$
$$= 11,300$$

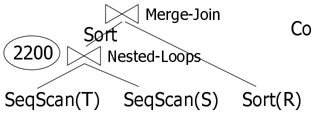
What is size of $S \bowtie T$?

$$|S \bowtie T| = \frac{|S|\cdot|T|}{\max\{V(S,B), V(T,B)\}} = 80,000$$

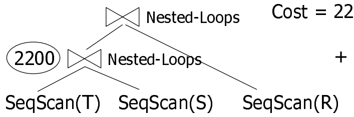
B(ST) = 80,000/50 = 1600

Level-2 Sub-Plans (cont.)

B(ST) = 1600

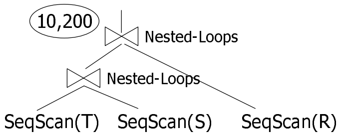


$$\text{Cost} = 3600 + B(ST) + 3\cdot(B(ST) + B(R)) = 10,900$$



$$\text{Cost} = 2200 + B(ST) + B(ST) + \left\lceil\frac{B(ST)}{M-2}\right\rceil\cdot B(R) = 10,200$$

Winner



- Selected plan: C = 10,200
- Worse plan: C > 201,000
- Benefit from optimizer: 20x speedup!

Summary

- Query Optimization
  - Statistics
  - Intermediate result size estimation
  - Cost estimation
  - Logical transformations & heuristics
  - Generating and comparing plans

Readings in Textbook

- Chapter 7, except:
  - 7.6, 7.7.2, 7.7.3, 7.7.4, 7.7.5, 7.7.6
  - Material on duplicate elimination operator, grouping, aggregation operators
- Note: The dynamic programming approach to enumeration and pruning is not included in reading list, but…
- You are responsible for understanding that material as covered in class!!!