Query Optimization in
Relational Database Systems

It is safer to accept any chance that offers itself, and extemporize a procedure to fit it, than to get a good plan matured, and wait for a chance of using it.

Thomas Hardy (1874)
in *Far from the Madding Crowd*

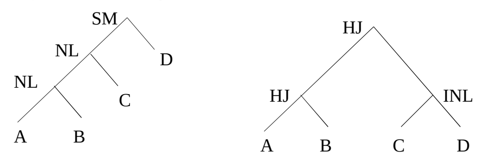
Review: Case where index is useful

Query Optimization

- Since each relational op returns a relation, ops can be *composed*!
- Queries that require multiple ops to be composed may be composed in different ways - thus *optimization* is necessary for good performance, e.g. $A \bowtie B \bowtie C \bowtie D$ can be evaluated as follows:
  - $(((A \bowtie B) \bowtie C) \bowtie D)$
  - $((A \bowtie B) \bowtie (C \bowtie D))$
  - $((B \bowtie A) \bowtie (D \bowtie C))$
  - …

Query Optimization

- Each strategy can be represented as a query evaluation plan (QEP) - Tree of R.A. ops, with choice of algo for each op.



- Goal of optimization: To find the “best” plan that compute the same answer (to avoid “bad” plans)

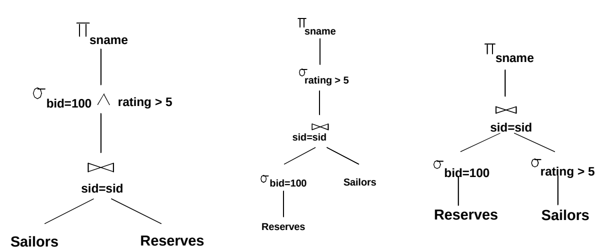
More on Motivating Examples

Sailors (<u>*sid*: integer</u>, *sname*: string, *rating*: integer, *age*: real)
Reserves (<u>*sid*: integer, *bid*: integer, *day*: dates</u>, *rname*: string)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

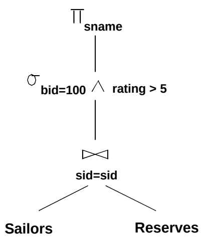
Example

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```



SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example



sname
lubber

sid	sname	rating	age	bid	day	rname
31	lubber	8	55.5	100	10/11/96	lubber

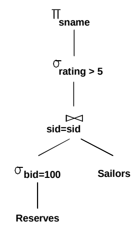
sid	sname	rating	age	bid	day	rname
31	lubber	8	55.5	100	10/11/96	lubber
58	rusty	10	35.0	103	11/12/96	dustin

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
31	100	10/11/96	lubber
58	103	11/12/96	dustin

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example



sname
lubber

sid	sname	rating	age	bid	day	rname
31	lubber	8	55.5	100	10/11/96	lubber

sid	sname	rating	age	bid	day	rname
31	lubber	8	55.5	100	10/11/96	lubber

sid	bid	day	rname
31	100	10/11/96	lubber

sid	bid	day	rname
31	100	10/11/96	lubber
58	103	11/12/96	dustin

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example

sname
lubber

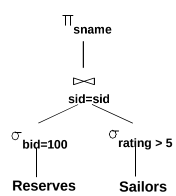
sid	sname	rating	age	bid	day	rname
31	lubber	8	55.5	100	10/11/96	lubber

sid	bid	day	rname
31	100	10/11/96	lubber

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
31	100	10/11/96	lubber
58	103	11/12/96	dustin

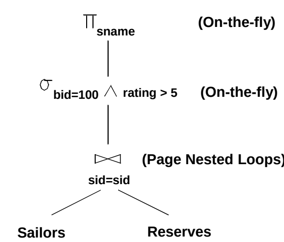


SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example (Cont)

- Cost?

Query Evaluation Plan:

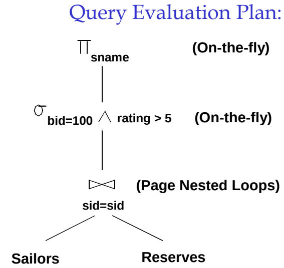


SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example (Cont)

- Cost: 500+500*1000 I/Os

Query Evaluation Plan:

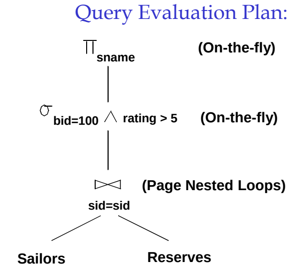


SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

Example (Cont)

- Cost: 500+500*1000 I/Os
- Memory?

Query Evaluation Plan:

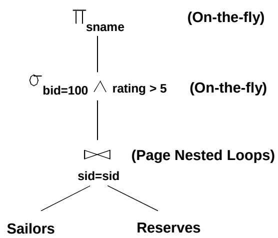


Example (Cont)

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

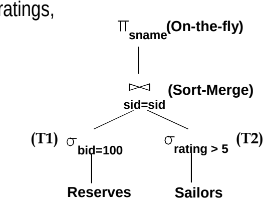
- Cost: 500+500*1000 I/Os
- Memory: 3

Query Evaluation Plan:



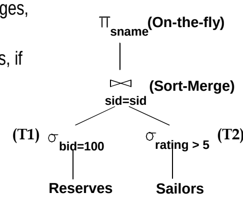
Alternative Plans 1 (No Indexes)

- Main difference: push selections down
- Assume 5 buffers, T1 = 10 pages (100 boats, uniform distribution), T2 = 250 pages (10 ratings, uniform distribution)



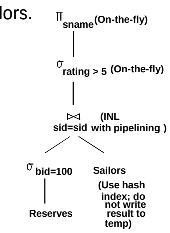
Alternative Plans 1 (No Indexes)

- Main difference: push selections down
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250)
  - Total: 4060 page I/Os.



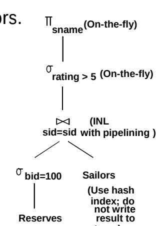
Alternative Plans 2 (With Indexes)

- Clustered index on bid of Reserves
  - 100,000/100 = 1000 tuples on 1000/100 = 10 pages
- Hash index on sid. Join column sid is a key for Sailors.
- INL with pipelining (outer is not materialized)
  - Project out unnecessary fields from outer doesn't help.
- At most one matching tuple, unclustered index on sid OK.
- Did not push "rating>5" before the join. Why?



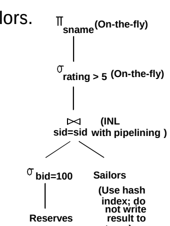
Alternative Plans 2 (With Indexes)

- Clustered index on bid of Reserves
  - 100,000/100 = 1000 tuples on 1000/100 = 10 pages
- Hash index on sid. Join column sid is a key for Sailors.
- INL with pipelining (outer is not materialized)
  - Project out unnecessary fields from outer doesn't help.
- At most one matching tuple, unclustered index on sid OK.
- Decision not to push rating>5 before the join is based on availability of sid index on Sailors.
- Cost?

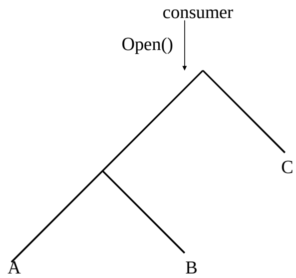


Alternative Plans 2 (With Indexes)

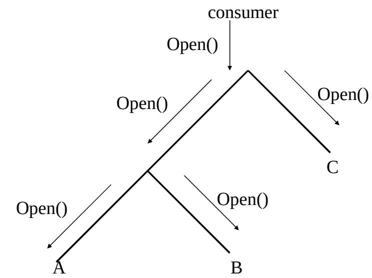
- Clustered index on bid of Reserves
  - 100,000/100 = 1000 tuples on 1000/100 = 10 pages
- Hash index on sid. Join column sid is a key for Sailors.
- INL with pipelining (outer is not materialized)
  - Project out unnecessary fields from outer doesn't help.
- At most one matching tuple, unclustered index on sid OK.
- Decision not to push rating>5 before the join is based on availability of sid index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple (1000*2.2); total 2210 I/Os.



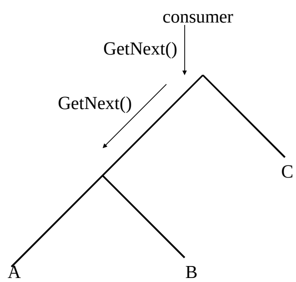
Plan Execution under the Iterator Model



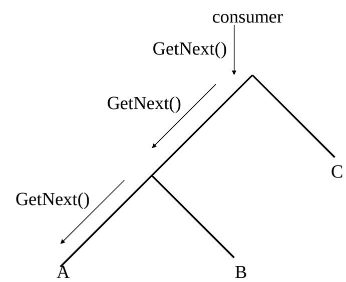
Plan Execution under the Iterator Model



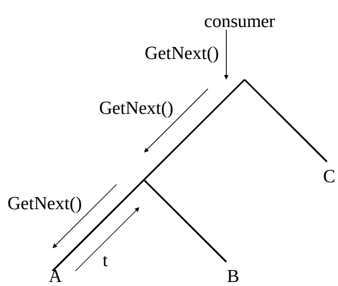
Plan Execution under the Iterator Model



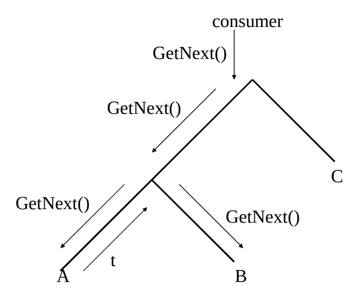
Plan Execution under the Iterator Model



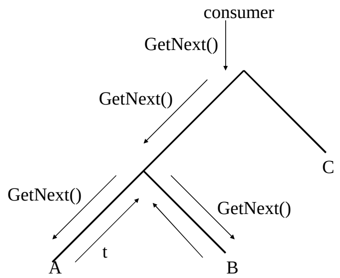
Plan Execution under the Iterator Model



Plan Execution under the Iterator Model

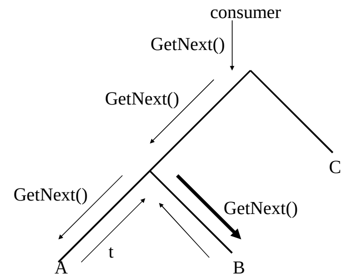


Plan Execution under the Iterator Model



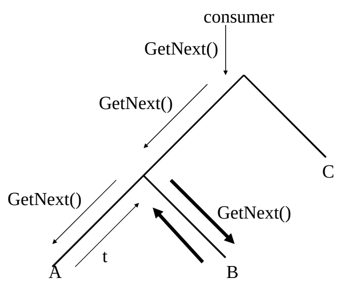
CS5208: Query Optimization 25

Plan Execution under the Iterator Model



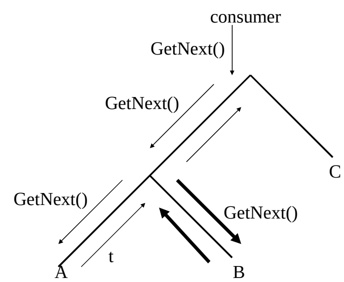
CS5208: Query Optimization 26

Plan Execution under the Iterator Model



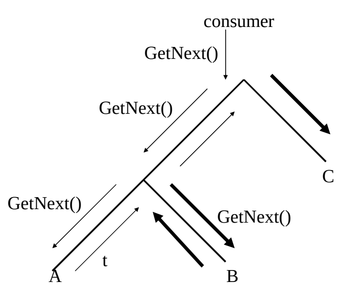
CS5208: Query Optimization 27

Plan Execution under the Iterator Model



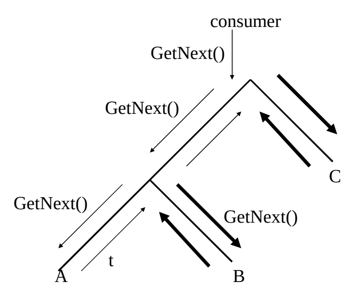
CS5208: Query Optimization 28

Plan Execution under the Iterator Model



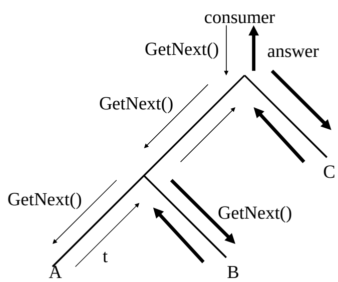
CS5208: Query Optimization 29

Plan Execution under the Iterator Model

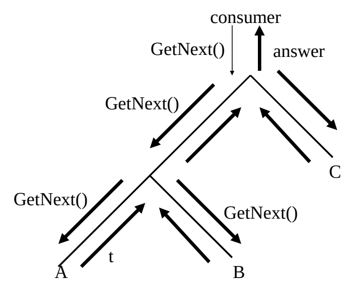


CS5208: Query Optimization 30

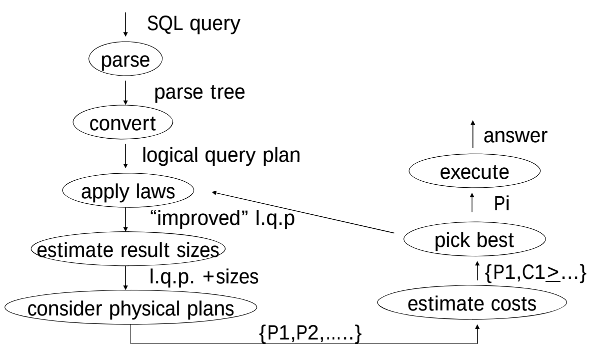
Plan Execution under the Iterator Model



Plan Execution under the Iterator Model



Overview of Query Optimization

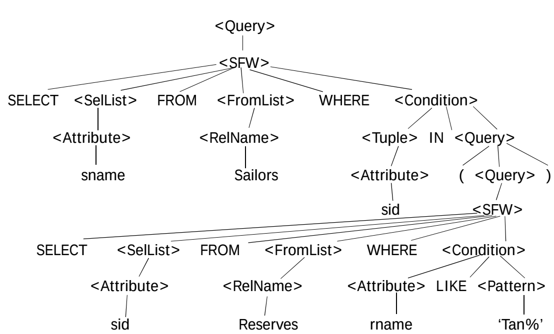


Example: SQL query

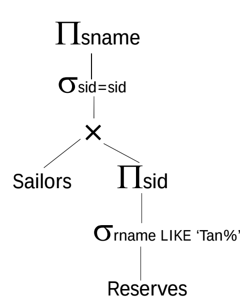
SELECT sname
FROM Sailors
WHERE sid IN (
    SELECT sid
    FROM Reserves
    WHERE rname LIKE 'Tan%'
);

(Find names of sailors whose reservation is made by someone whose name begins with "Tan")

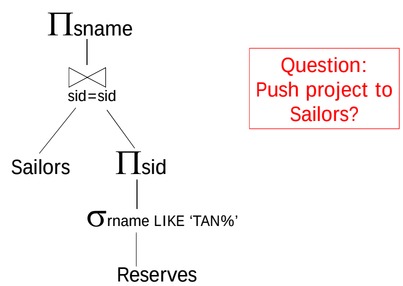
Example: Parse Tree



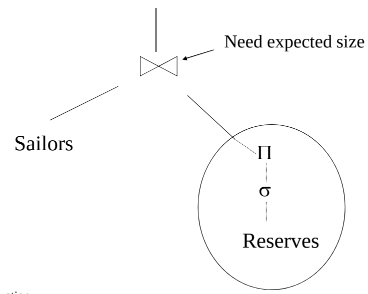
Example: Logical Query Plan



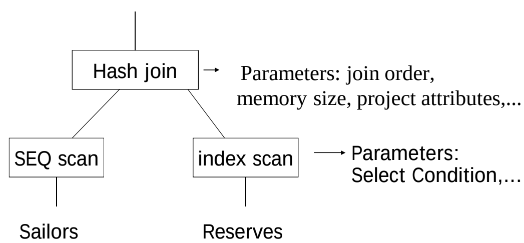
Example: Improved Logical Query Plan



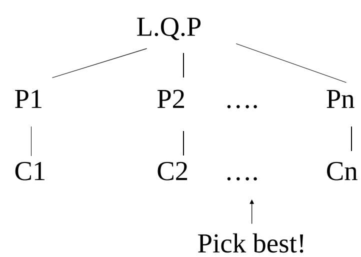
Example: Estimate Result Sizes



Example: One Physical Plan



Example: Estimate costs



Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- Rules on joins, cross products and union

  $R \bowtie S = S \bowtie R$

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

Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- Rules on joins, cross products and union

  $R \bowtie S = S \bowtie R$

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

  $R \times S = S \times R$

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

Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- Rules on joins, cross products and union

$R \bowtie S = S \bowtie R$

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

$R \times S = S \times R$

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

$R \cup S = S \cup R$

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

Rules: Selects

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

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

Rules: Selects

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

$\sigma_{p1 \vee p2}(R) = [\sigma_{p1}(R)] \cup [\sigma_{p2}(R)]$

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

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

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

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

Rules: Project

Let: X = set of attributes

Y = set of attributes

XY = X U Y

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

$\pi_x(R) = \pi_x [\pi_y (R)]$ if y contains x

Rules: $\sigma + \bowtie$ combined

Let P = predicate with only R attribs
Q = predicate with only S attribs
M = predicate with only R,S attribs

$\sigma_p (R \bowtie S) =$

$\sigma_q (R \bowtie S) =$

Rules: $\sigma + \bowtie$ combined

Let P = predicate with only R attribs
Q = predicate with only S attribs
M = predicate with only R,S attribs

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

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

Bags vs. Sets

R = {a,a,b,b,b,c}
S = {b,b,c,c,d}
RUS = ?

- Option 1 SUM
  RUS = {a,a,b,b,b,b,b,c,c,c,d}
- Option 2 MAX
  RUS = {a,a,b,b,b,c,c,d}

"SUM" is implemented

- Use "SUM" option for bag unions
- Some rules cannot be used for bags
  - e.g. $A \cap_s (B \cup_s C) = (A \cap_s B) \cup_s (A \cap_s C)$

Let A, B and C be {x}
$B \cup_B C = \{x, x\}$ **$A \cap_B (B \cup_B C) = \{x\}$**
$A \cap_B B = \{x\}$ $A \cap_B C = \{x\}$
$(A \cap_B B) \cup_B (A \cap_B C) = \{x, x\}$

Review

- Consider the join R JOIN(R.a=S.b) S, given the following information about the relations to be joined. The cost metric is the number of page I/Os, and the cost of writing out the result should be ignored.
  - R contains 10,000 tuples and has 10 tuples per page.
  - S contains 20,000 tuples and has 10 tuples per page.
  - S.b is the primary key for S.
  - Both relations are stored as simple heap files.
  - 102 buffer pages are available (inclusive of input/output buffers).
- What is the cost of joining R and S using a block nested-loops join algorithm? What is the minimum number of buffer pages required for this cost to remain unchanged?
- What is the cost of joining R and S using a sort-merge join algorithm? What is the minimum number of buffer pages required for this cost to remain unchanged?

Review

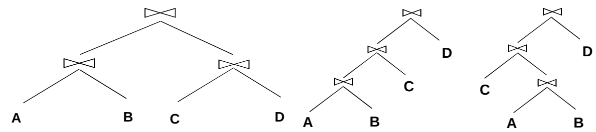
- Block Nested Loops Join.
  - Using R as the outer relation, and 1 page for input and output buffer.
  - cost = 10,000/10 + (10,000/10)/100*20,000/10 = 21,000
  - minimum number of buffer page s= 102 (no change)
- Sort-merge Join
  - Each relation needs 2 passes to sort
  - Cost to sort R = 2*2*10,000/10; cost to sort S = 2*2*20,000/10
  - Cost = 4000+8000+1000+2000 = 15,000
  - min buffer required is the same as that required to sort the larger relation, which is S. So, min buffer = 46

Query Optimizer

- Find the "best" plan (more often avoids the bad plan)
- Comprises the following
  - Plan space
    - huge number of alternative, semantically equivalent plans
    - computationally expensive to examine all
    - Conventional wisdom: avoid bad plans
      - need to include plans that have low cost
  - Cost model
    - facilitate comparisons of alternative plans
    - has to be "accurate"
  - Enumeration algorithm (Search space)
    - search strategy (optimization algorithm) that searches through the plan space
    - has to be efficient (low optimization overhead)

Plan Space

- Left-deep trees: right child has to be a base table
- Right-deep trees: left child has to be a base table
- Deep trees: one of the two children is a base table
- Bushy tree: unrestricted



Bushy tree   Left-deep tree   Deep tree

Cost Models

- Typically, a combination of CPU and I/O costs
- Objective is to be able to rank plans
  - exact value is not necessary
- Relies on
  - statistics on relations and indexes
  - formulas to estimate CPU and I/O cost
  - formulas to estimate selectivities of operators and intermediate results

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must estimate size of result for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assuming independence of predicates can simplify size estimation but is error prone.

Statistics and Catalogs

- Need information about the relations and indexes involved. Catalogs typically contain at least:
  - # tuples of R (T(R)), #bytes in each R tuple (S(R))
  - # blocks to hold all R tuples (B(R))
  - # distinct values in R for attribute A (V(R,A))
  - NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.

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 string
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 estimate for W = $\sigma_{Z=val}$ (R)

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

$$T(W) = \frac{T(R)}{V(R,Z)}$$

S(W) = S(R)

Assumption:

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

Alternative assumption: use DOM(R,Z)

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

Solution: Estimate values in range

R

	Z

Min=1
Max=20

V(R,Z)=10

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

$$f \text{ (fraction of range)} = \frac{20\text{-}15\text{+}1}{20\text{-}1\text{+}1} = \frac{6}{20} \qquad T(W) = f \times T(R)$$

Alternative: (Max(Z)-value)/(Max(Z)-Min(Z))

W = R1 ⋈ R2

R

A	B	C

S

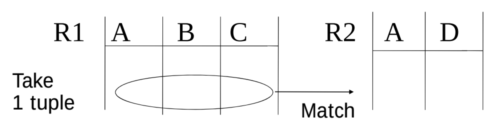
A	D

Assumption:

V(R1,A) ≤ V(R2,A) ⇒ Every A value in R1 is in R2
V(R2,A) ≤ V(R1,A) ⇒ Every A value in R2 is in R1

"containment of value sets"

Computing T(W) when V(R1,A) ≤ V(R2,A)



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

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

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

For complex expressions, need intermediate T,S,V results.

E.g. W = [$\sigma_{A=a}$ (R1)] ⋈ R2

Treat as relation U

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

Also need V (U, *) !!

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) = ?

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) = ?

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(R,B)) V(U,C) =

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 V(U,B) and V(U,C)

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)

(Assumption: Preservation of value sets)

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: $U = R1 \bowtie R2$

$T(U) = \frac{1000 \times 2000}{200}$

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

Z = U $\bowtie$ *R3*

$T(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300}$

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

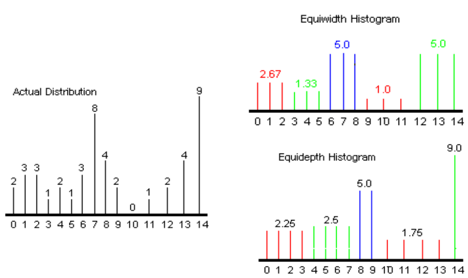
Estimating Size of Plan

- Since a plan may contain multiple operators, need to propagate statistical information to those operators.
- Errors
  - source include uniformity assumption, and inability to capture correlation
  - propagated to other operators at the higher level of the plan tree
- During runtime, may need to sample the actual intermediate results
  - *dynamic* query optimization

Statistical Summaries of Data

- More detailed information are sometimes stored e.g., histograms of the values in some field
  - a histogram divides the values on a column into k buckets
    - k is predetermined or computed based on space allocation.
  - several choices for "bucketization" of values
    - If a table has n records, an equi-depth histograms divides the set of values on a column into k ranges such that each range has the same number of records, i.e., n/k.
    - Equi-width histograms.
    - Frequently occurring values may be placed in singleton buckets.
  - histograms on single column do not provide information on the correlations among columns
    - 2-dimensional histograms can be used but too many buckets!

Histograms



Search Algorithms

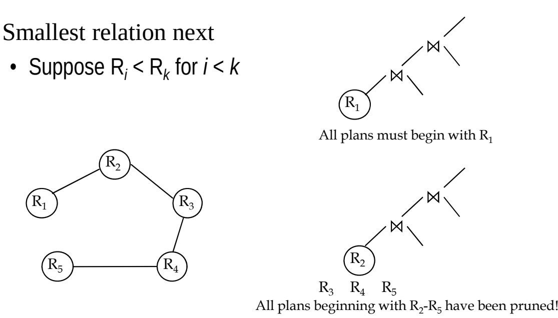
- Exhaustive
  - enumerate each possible plan, and pick the best
- Greedy Techniques
  - smallest relation next
  - smallest result next
  - typically polynomial time complexity
- Randomized/Transformation Techniques
- System R approach
  - Dynamic Programming with Pruning

Multi-Join Queries

- Focus on multi-join queries first
  - Join is the most expensive operations
  - Selections and projections can be pushed down as early as possible
- Query
  - a query graph whose nodes are relations and edges represent a join condition between the two nodes

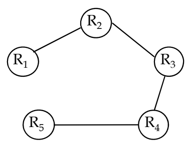
Greedy Algorithm (Example)

- Smallest relation next
  - Suppose $R_i < R_k$ for $i < k$



Greedy Algorithm (Example)

- Smallest relation next
  - What if R1 < R5 < R3 < R2 < R4???



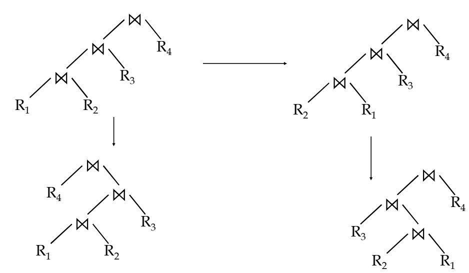
Randomized Techniques

- Employ randomized/transformation techniques for query optimization
- State space -- space of plans, State -- plan
- Each state has a cost associated with it
  - determined by some cost model
- A move is a perturbation applied to a state to get to another state
  - a move set is the set of moves available to go from one state to another
  - any one move is chosen from this move set randomly
  - each move set has a probability associated to indicate the probability of selecting the move

More on Randomized Techniques

- Two states are neighboring states if one move suffices to go from one state to the other
- A local minimum in the state space is a state such that its cost is lower than that of all neighboring states
- A global minimum is a state which has the lowest cost among all local minima
  - at most one global minimum
- A move that takes one state to another state with a lower cost is called a downward move; otherwise it is an upward move
  - in a local/global minimum, all moves are upward moves

Randomized Algorithm (Example)



Local Optimization

```
S = initialize()
minS = S
repeat {
   repeat {
      newS = move(S)
      if (cost(newS) < cost(S))
         S = newS
   } until ("local minimum reached")
   if (cost(S) < cost(minS))
      minS = S
   newStart(S);
} until ("stopping condition satisfied")
return (minS);
```

A move is accepted if the adjacent state being moved to has a lower cost

By doing so repeatedly, a local minimum can be reached

Run: sequence of moves to a local minimum from the start state

Issues on Local Optimization

- How is the start state obtained?
  - The state in which we start a run.
  - The start state of the first run is the initial state.
  - All start states should be different.
  - Should be obtained quickly
    - random
    - greedy heuristics
    - making a number of moves from the local minimum, except that this time each move is accepted irrespective of whether it increases or decreases the cost
- How is the local minimum detected?
- How is the stopping criterion detected?

Issues on Local Optimization (Cont)

- How is the local minimum detected?
  - Not practical to examine all neighbors to verify that one has reached a local minimum.
  - Based on random sampling
    - examine a sufficiently large number of neighbors
    - if any one is lower, we move to that state, and repeat the process
    - if no tested neighbor is of lower cost, the current state can be considered a local minimum
    - the number of neighbors to examine can be specified as a parameter, and is called the sequence length.

Issues in Local Optimization (Cont)

- How is the stopping criterion detected?
  - Determines the number of times that the outer loop is executed.
  - Can be fixed and is given by sizeFactor*N, where sizeFactor is a parameter, N is the number of relations.

Transformation Rules

- Restricted to left-deep trees
  - all possible permutations of the N relations
  - let S be the current state, S = (... i ... j ... k ...)
  - swap
    - select two relations, say i and j at random. Check if interchanging them results in a valid permutation. If so, the move consists of swapping i and j to get the new state newS = (... j ... i ... k ...)
  - 3Cycle
    - select three relations, say i and j and k at random. The move consists of cycling i, j and k: i is moved to the position of j, j is moved to the position of k and k is moved to the position of i. Check if resulting permutation is valid. If so, the move consists of swapping i and j to get the new state newS = (... k ... i ... j ...)
- Other methods (e.g., join methods)? Bushy trees?

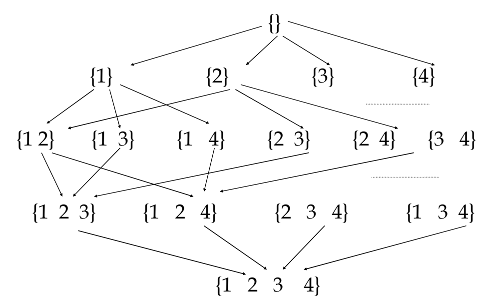
Comparison between Exhaustive, Greedy and Randomized Algorithms

- Plan quality
- Optimization overhead

Dynamic Programming (Left-Deep Trees)

- The algorithm proceeds by considering increasingly larger subsets of the set of all relations.
- Plans for a set of cardinality *i* are constructed as extensions of the best plan for a set of cardinality *i*-1
- Search space can be pruned based on the principal of optimality
  - if two plans differ only in a subplan, then the plan with the better subplan is also the better plan

Dynamic Programming (Cont)



Dynamic Programming (Left-Deep Trees)

- accessPlan(R) produces the best plan for relation R
- joinPlan(p1,R) extends the join plan p1 into another plan p2 in which the result of p1 is joined with R in the best possible way
- Optimal plans for subsets are stored in optplan() array and are reused rather than recomputed

Dynamic Programming (Cont)

```
for i = 1 to N
    optPlan({Ri}) = accessPlan(Ri)
for i = 2 to N {
    forall S subset of {R1, R2, … Rn} such that |S|=i {
            bestPlan = dummy plan with infinite cost
            forall Rj, Sj such that S = {Rj} U Sj {
                    p = joinPlan(optPlan(Sj), Rj)
                    if cost(p) < cost(bestPlan)
                        bestPlan = p
            }
            optPlan(S) = bestPlan
    }
}
Popt = optPlan{R1, R2, … Rn}
```


Dynamic Programming Example

Consider the join of 4 relations, R, S, T and U
Each table has 1000 tuples
Assume intermediate result size (tuples) as cost metrics

R(a,b)	S(b,c)	T(c,d)	U(d,a)
V(R,a)=100			V(U,a)=50
V(R,b)=200	V(S,b)=100		
	V(S,c)=500	V(T,c)=20	
		V(T,d)=50	V(U,d)=1000

Example (Cont)

	{R}	{S}	{T}	{U}
Size	1,000	1,000	1,000	1,000
Cost	0	0	0	0
BestPlan	R	S	T	U

Example (Cont)

	{R,S}	{R,T}	{R,U}	{S,T}	{S,U}	{T,U}
Size	5,000	1M	10,000	2,000	1M	1,000
Cost	0	0	0	0	0	0
BestPlan	R ⋈ S	R ⋈ T	R ⋈ U	S ⋈ T	S ⋈ U	T ⋈ U

What about S ⋈ R since its cost is also 0??

Example (Cont)

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
Size	10,000	50,000	10,000	2,000
Cost	2,000	5,000	1,000	1,000
BestPlan	(S ⋈ T) ⋈ R	(R ⋈ S) ⋈ U	(T ⋈ U) ⋈ R	(T ⋈ U) ⋈ S

Example (Cont)

Grouping	Cost
((S ⋈ T) ⋈ R) ⋈ U)	12,000
((R ⋈ S) ⋈ U) ⋈ T)	55,000
((T ⋈ U) ⋈ R) ⋈ S)	11,000
((T ⋈ U) ⋈ S) ⋈ R)	3,000
(T ⋈ U) ⋈ (R ⋈ S)	6,000
(R ⋈ T) ⋈ (S ⋈ U)	2M
(S ⋈ T) ⋈ (R ⋈ U)	12,000

Example (Cont)

Grouping	Cost
((S ⋈ T) ⋈ R) ⋈ U)	12,000
((R ⋈ S) ⋈ U) ⋈ T)	55,000
((T ⋈ U) ⋈ R) ⋈ S)	11,000
((T ⋈ U) ⋈ S) ⋈ R)	**3,000**
(T ⋈ U) ⋈ (R ⋈ S)	6,000
(R ⋈ T) ⋈ (S ⋈ U)	2M
(S ⋈ T) ⋈ (R ⋈ U)	12,000

Dynamic Programming (Cont)

- Time & Space complexity
  - For k relations, for left-deep trees, $2^k - 1$ entries!
  - For bushy trees, $O(3^k)$
- DP may maintain multiple plans per subset of relations
  - Interesting orders
- Is DP optimal?

Summary

- Query optimization is NP-hard.
- Instead of finding the best, the objective is largely to avoid the bad plans
- Many different optimization strategies have been proposed
  - greedy heuristics are fast but may generate plans that are far from optimal
  - dynamic programming is effective at the expense of high optimization overhead