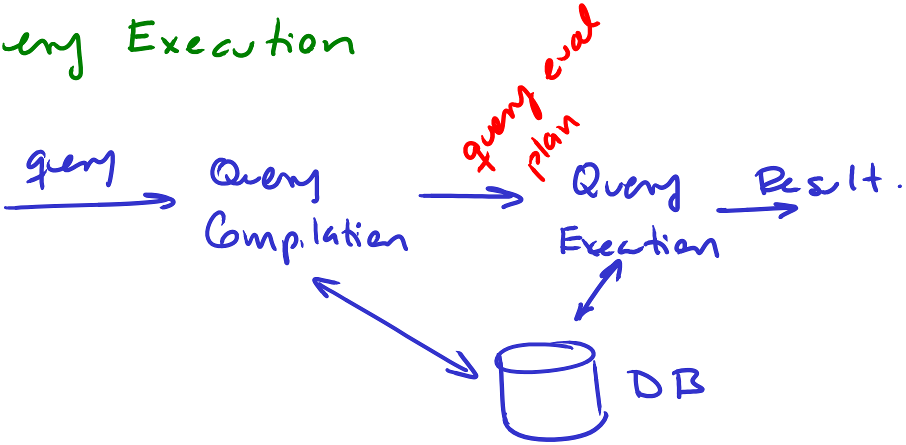
Query Execution



Query Compilation

a) Parsing. A parse tree is constructed
- Create an algebraic expression.

b) Query Rewrite:
- Several equivalent query expression

c) Physical plan generation
- Each expression is converted to an evaluation plan by indicating the alg. to use.

b) and c) are the query optimizer

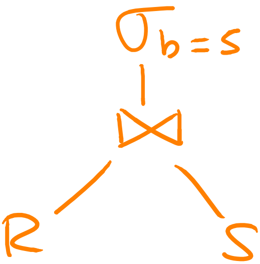
⇒ find best query plan:

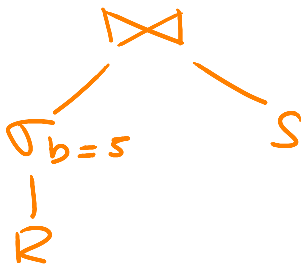
1) Which algebraic expression is the one leading to the most efficient alg.
2) For each operation in the expression which alg. will be used to answer it.
3) How should each operation pass data to the next operation.
4) How are the relations going to be accessed.

Ex: $R(a,b)$ $S(a,c)$

SELECT * from R natural Join S
WHERE $b=5$

Equivalent Expressions





Annotate tree with algorithms and access methods

alg. to use.

on-the fly

on the fly

Result is not stored to disk!!

Nested Loop Join

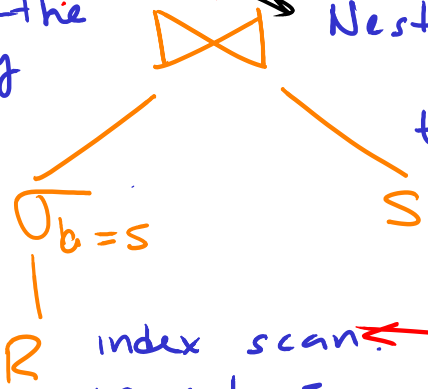


table scan

index scan using $b=5$.

bottom of tree

Access to relations.

Estimate cost.

$\Rightarrow$ choose fastest!

Access to tuples:

- Sequential scan of heap of Rel.

or

- Using an index to scan a subset of tuples of R (index scan)

Result of query:

- kept in memory.

Iterators:

- Many operations access only one tuple at a time.
  - read tuple.
  - inspect
  - dispose
  - read next tuple. .

Open() — initiates the process

GetNext() — return next tuple

close() – ends process

Example:

$$\pi_a \, \sigma_{b=3} \, R$$

π_a on the fly.

$\sigma_{b=3}$ on the fly.

seq scan of R

R

π and σ can be implemented as iterators

σ inspects one tuple at a time, sends one tuple at a time to π

No need to store any tuple in memory

Parameters to measure cost

M. Amount of memory available in number of blocks

B(R) # of blocks used by heap of R

|R| # of tuples of R (book uses T(R))

V(R, a) # of different values of att a in R

In general:

$$V(R, [a_1, a_2 \dots a_n]) = |\gamma_{a_1, a_2 \dots a_n} R|$$

$\Rightarrow$ # of different values for tuple $a_1 \dots a_n$

Cost Model

- We assume that the major component of cost is I/O
- Cost of read equal to cost of write
- Cost of random access of pages equal to cost of seq access.

Algorithms to answer queries.

2 main classifications.

a) based on type of algorithm:

1) Sorting based
2) Hash based
3) Index based

b) based on difficulty.

1) One-pass: Relations are read only once.
2) Two passes.
   - Read data (1st pass)
   - Process.
   - Write data.
   - Read data again. (2nd pass).

2nd pass might read diff number of blocks than 1st pass.

3) Three or more passes.
(needed for very large relations).
   - Generalization of Two passes.

One Pass Alg.

1) Tuple-at-a time π, σ

- We can read one block at a time.

  $\Rightarrow$ use <u>one</u> memory buffer.

π_a
|
R

- Read one block at a time,
- Inspect each tuple, output result
- Repeat.

or

if we received tuples from another operation, one tuple at a time with no need for buffering.

(on the fly $\equiv$ no memory needed)

π_a.
|
$\bowtie$

on the fly.

- Receive tuple from $\bowtie$ via iterator.
- Output result
- Repeat.

No block in memory needed.

But assume 1 block for simplicity's sake.

Other one pass unary operators.

Duplicate elimination (δ)

- Read each tuple.
  - If we have seen it, ignore
  - Otherwise output and keep track of it.

We need to keep a copy of each distinct tuple.

input tuples (iterator or from R heap)

at most M - 1 available for distinct.

We do not need block for output.

$\Rightarrow$ tuples in result output immediately.

We can do δR in one pass as long as:

$$B(\delta(R)) < M$$

Book uses:

$$B(\delta(R)) \leq M$$

because $M \gg 1$

But, how do we know $B(\delta(R))$ without calculating $\delta(R)$ first?

$\Rightarrow$ Statr.

$R(a_1, a_2 \dots a_n)$

then.

We can use $V(R, a_1 \dots a_n)$ and the size of the tuple in R to calculate $\delta(R)$.

Group By:

Generalization of $\delta(R)$

Remember

$$\delta(R) = \gamma^{a_1 \dots a_n} R$$

For $\gamma^{\langle att\,list \rangle}_{\langle explist \rangle} R$.

We need to keep track of:

- Each different value of <attlist>.
- Info needed to compute <explist>.

- min(x) } Keep current min/max
  max(x)
- sum(x) · Keep current sum
- count(x) Keep current count
- avg(x) Keep both current count and sum.

We cannot output tuples until we have read all input tuples.

- We must also create access structures in memory (hash tables, b+trees) to efficiently find group tuple belongs to.

- In general
  - The amount of memory required per group is small.
  - Proportional to the number of different groups.

$$|\gamma_{\langle explist \rangle}^{a_1 \dots a_i} R| \propto V(R, a_1 \dots a_i)$$

Not a lot of memory required per tuple in addition to $a_1 \dots a_i$

We can do it in one pass if we have enough memory to

- hold all different groups
- data structures for quick access to groups.
- any data required to compute grouping function.

In general size of tuple of result much smaller than original tuple.

So we simplify

We can do group-by in one pass if

$$B(\gamma^{a_1 \dots a_j} R)$$

hard to approximate

or $B(\delta(\Pi_{a_1 \dots a_j} R)) < M$.

which is based on evaluating the size of δ

One Pass alg. for binary operations:

$\cup, \cap, -, \times, \bowtie$

In practice set operations of two types:

- True sets: No duplicates (default).
- Bags: duplicates.

UNION
INTERSECT } ALL
EXCEPT

$\Rightarrow$ Represented $\cup_B, \cap_B, -_B$

TABLE R UNION ALL TABLE S

Result contains all tuples in R plus all tuples in S.

TABLE R INTERSECT ALL TABLE S

if a tuple in has m duplicates in R and n duplicates in S result contains min(m,n) duplicates of tuple.

TABLE R EXCEPT ALL TABLE S

if a tuple in has m duplicates in R and n duplicates in S result contains min(m−n, ∅)

$\cup_B$

- Similar to π:
  - We only need to inspect one tuple at a time.

    $M = 1$. regardless of size of input. Read one relation at a time

$\cup$

- Removes duplicates;
- Equivalent to. $\delta(R \cup_B S)$

The book is wrong. It states we only need to read S in $M-1$ and do one-tuple-at-a-time for R (page 716)

We can do in one pass if

$$\delta(\underbrace{R \cup_B S}_{\text{hard to estimate!}}) \leq M$$

Instead, we can approximate to:

$$\delta(B(R)) + \delta(B(S)) \leq M$$

We can remove duplicates as we read tuples:

if tuple already read, ignore
otherwise { output
add to read tuples.

$\cap$, $\cap_B$, $\times$, $\bowtie$,

- All commutative operations.
- Keep smaller table in memory (plus data structures for fast access).
- Plus at most one block for other table:

One pass if, approximately:

$$\min(B(R), B(S)) \le M.$$

Specifically for each of these operations:

Because they are commutative, assume

$$B(R) \geq B(S)$$

Use M-1 blocks for data in S

Use 1 block to read R

$\cap$, $\cap_B$

Read S, organize in data structure.

Remove duplicates!

for every tuple t in R

if t in S

if bag op $\Rightarrow$ output t if needed

otherwise output t first time only.

We only need; approximately

$$B(\delta(S)) < M.$$

a bit more to / keep counter for $\cap_B$

×

Read S

for every block of R

for every tuple t in this block

for every tuple s in S

compute cross product, output.

already in Memory

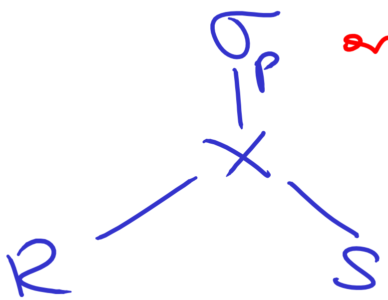
Requires $B(S) < M$

1 block for R.

$R \bowtie_p S$ Join is a special case of cross product.

$\sigma_p (R \times S)$.

σ_p Can be done on the fly.



on the fly (does not need memory).

But the DBMS will do it both in one operation.

Identical to X, with extra condition

Read S

for every block of R

for every tuple t in this block

for every tuple s in S

if t and s satisfy P

output join(t, s)

$S - R$, $S -_B R$ $B(S) < B(R)$

We need to remove duplicates of S

Load S,

Remove duplicates. for $-_B$ keep count of each tuple.

Scan R one block at a time

- for every tuple in R,

if in S

for $-$

mark as not in result

for $-_B$

subtract one from count.

Output result

Required memory: $B(\delta(S)) < M$.

(slightly more for data structures and data) counts). plus 1 block for R

We always read smaller table into M

To compute $R - S$, $R -_B S$.

Read S
for every tuple t in R
if t not in S
output
(for $-$ also keep track of those output)

To compute $S - R$, $S -_B R$

Read S
for $-$ remove all duplicates at the same time.
For every tuple t in R
if t in S
remove from S
for $-_B$ remove one duplicate only
Output tuples left in S

Again, I think the book is wrong because it does not know how to properly compute $R - S$. and $R -_B S$.

It assumes R has no duplicates!

Summary of 1 pass algorithm.

	Approx blocks of M required
$\pi, \sigma, \cup_B$	1
$\gamma\ \delta$	$\delta(B(R)) < M$
$\cup$	$\delta(B(R)) + \delta(B(S)) < M$
$\cap, \cap_B$	$\min(\delta(B(R)), \delta(B(S))) < M$
$\bowtie, \times$	$\min(B(R), B(S)) < M$
$S - R$, $S -_B R$	$\delta(B(S)) < M$

order by is a variation of γ, δ
denoted τ

Block based Nested Join.

Generalization of 1 pass join.

- What if no relation fits in memory?

  Assume: $B(S) > M$.

  outside loop. $B(R) > M$

For each M-1 blocks of S
   Read blocks and organize them in mem.
   For each block of R.
      for every tuple r in R
         find matching tuples in read blocks of S.

inside loop.

Each block of R is read

$$\left\lceil \frac{B(S)}{M} \right\rceil \text{ times}$$

We also need to read S: B(S).

Total cost:

$$\left\lceil \frac{B(S)}{M} \right\rceil \cdot B(R) + B(S)$$

To minimize, make outside table the smallest!

Because tables are usally large we approximate to:

Cost:

$$B(R) \cdot \left\lceil \frac{B(S)}{M} \right\rceil + B(S) \cong \frac{B(R) \cdot B(S)}{M}.$$

We should still read smallest table in the outside loop, but that cost might be neglegible

This alg. is usally worse than sort-merge join.

Two pass algorithms based on sorting

Algorithms that read data twice.

- Read tuples
- Process tuples
- Write tuples to disk

} first pass.

- Read tuples
- Process tuples.
  ⇒ output result

} second pass.

Sorting τ

- By sorting we can implement other operations (e.g. $\cap, \gamma, \bowtie$).

Two Phase Multiway Merge Sort TPMMS

- Alg. to sort large relations.

  $B(R) > M$

- Phase 1:

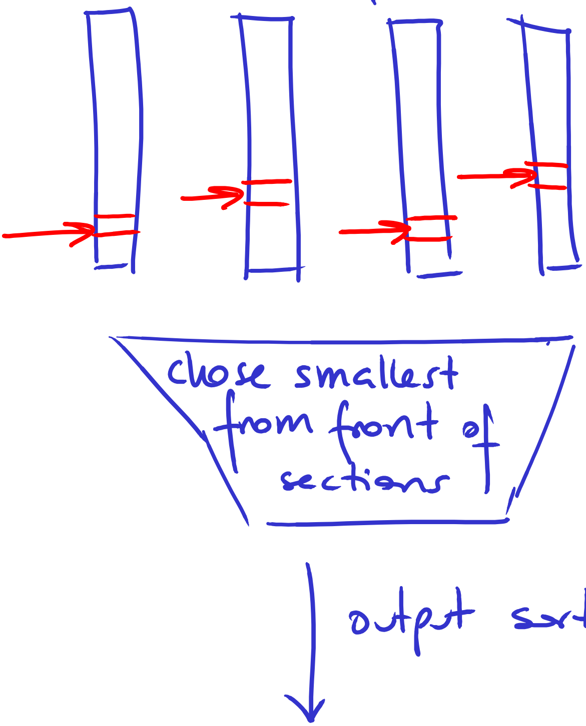
  For each M blocks of R :
    Read M blocks.
    Sort
    write back to temp. storage.

This creates $\left\lceil \frac{B(R)}{M} \right\rceil$ sorted sections

If # sorted sections $\leq M-1$
then

Phase 2:

- Merge sorted sections by reading one block of each section at a time.
- Use 1 block for output.



sorted sections of at most M-1 blocks.

if # sections $\geq M$ we might need 3 or more phases

Memory required

$$\left\lceil \frac{B(R)}{M} \right\rceil \leq M - 1$$

$\Rightarrow$ Approximately $B(R) \leq M^2$

Cost:

Phase 1: $B(R)$ Read $B(R)$ Write.

Phase 2: $B(R)$ Read

Assume cost of Read = Write

$\Rightarrow 3\,B(R)$

and output is sorted.

We can generalize # pases to.

$$\left\lceil \log_{M-1} B(R) \right\rceil \;..$$

But usually with a decent amount of memory we can sort very large relations in 2 passes.

$$B(R) < M^2$$

Duplicate elimination $\delta(R)$

- Sort R using TPMMS
- During second phase, output only first tuple of each set of duplicates

Mem required:

$$B(R) \leq M^2$$

Cost: $3B(R)$

Group By γ

Use TPMMS to sort by aggr. attributes

Like $\delta(R)$, during second phase

for each group of tuples in output
compute aggregation
output result

Requires one pass of tuples in group.
Memory required for computing agg. is less than 1 block.

Total mem required $B(R) \leq M^2$

Cost: $3B(R)$

$\cup, \cap, -$

We can also use TPMMS

- Do phase one of R
- Do phase one of S.
- Phase 2: do both R and S at the same time:

Read 1 block of each section of R and S at a time:

$\Rightarrow$ do operation on tuples in memory.

We need $\left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil < M$

for second pass:

$\Rightarrow$ Memory required is approx.

$$B(R) + B(S) \leq M^2$$

Cost: $3(B(R) + B(S))$