

CSE 562 Database Systems

Query Processing: Physical Operators

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson Prentice Hall 2nd Edition
©2008 Garcia-Molina, Ullman, and Widom

cse@buffalo

Outline – Query Optimization

- Overview
- Relational algebra level
 - Algebraic Transformations
- Detailed query plan level
 - Estimate Costs
 - Estimating size of results
 - Estimating # of IOs
 - Generate and compare plans

Algorithms for Algebra Operators

- Three primary techniques
 - Sorting
 - Hashing
 - Indexing
- Three degrees of difficulty
 - data small enough to fit in memory
 - too large to fit in main memory but small enough to be handled by a “two-pass” algorithm
 - so large that “two-pass” methods have to be generalized to “multi-pass” methods (quite unlikely nowadays)

Estimating IOs

- Count # of disk blocks that must be read (or written) to execute query plan

Additional Cost Estimation Parameters

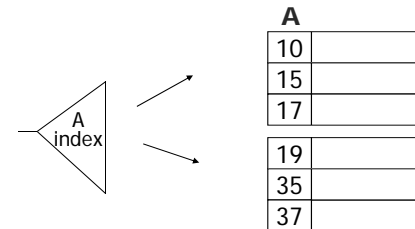
- $B(R)$ = # of blocks containing R tuples
- $f(R)$ = max # of tuples of R per block
- M = # memory blocks available
- $HT(i)$ = # levels in index i
- $LB(i)$ = # of leaf blocks in index i

UB CSE 562 Spring 2009

5

Clustering Index

- Index that allows tuples to be read in an order that corresponds to physical order



UB CSE 562 Spring 2009

6

Clustering Can Radically Change Cost

- Clustered file organization

R1 R2 S1 S2	R3 R4 S3 S4	...
-------------	-------------	-----

- Clustered relation

R1 R2 R3 R4	R5 R5 R7 R8	...
-------------	-------------	-----

- Clustering index

UB CSE 562 Spring 2009

7

Example

$R1 \bowtie R2$ over common attribute C

$T(R1) = 10,000$

$T(R2) = 5,000$

$S(R1) = S(R2) = 1/10$ block

Memory available = 101 blocks

→ Metric: # of IOs
(ignoring writing of result)

UB CSE 562 Spring 2009

8

Caution!

This may not be the best way to compare

- ignoring CPU costs
- ignoring timing
- ignoring double buffering requirements

Options

- Transformations: $R1 \bowtie R2$, $R2 \bowtie R1$
- Join algorithms:
 - Iteration (nested loops)
 - Merge join
 - Join with index
 - Hash join

Example

- **Iteration Join** (conceptually – without taking into account disk block issues)
 - for each $r \in R1$ do
 - for each $s \in R2$ do
 - if $r.C = s.C$ then output r,s pair

Example

- **Merge Join** (conceptually)
 - (1) if $R1$ and $R2$ not sorted, sort them
 - (2) $i \leftarrow 1$; $j \leftarrow 1$;
 - While $(i \leq T(R1)) \wedge (j \leq T(R2))$ do
 - if $R1\{i\}.C = R2\{j\}.C$ then *outputTuples*
 - else if $R1\{i\}.C > R2\{j\}.C$ then $j \leftarrow j+1$
 - else if $R1\{i\}.C < R2\{j\}.C$ then $i \leftarrow i+1$

Example

Procedure *outputTuples*

```
While (R1{ i }.C = R2{ j }.C) ∧ (i ≤ T(R1)) do
  [ jj ← j;
    while (R1{ i }.C = R2{ jj }.C) ∧ (jj ≤ T(R2)) do
      [ output pair R1{ i }, R2{ jj };
        jj ← jj+1 ]
    i ← i+1 ]
```

UB CSE 562 Spring 2009

13

Example

i	R1{i}.C	R2{j}.C	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

UB CSE 562 Spring 2009

14

Example

- **Join with Index** (Conceptually)

```
For each r ∈ R1 do
  [ X ← index (R2, C, r.C)
    for each s ∈ X do
      output r,s pair ]
```

Assume R2.C index

Note: X ← index(rel, attr, value)
then X = set of rel tuples with attr = value

UB CSE 562 Spring 2009

15

Example

- **Hash Join** (Conceptual)
 - Hash function h, range 0 → k
 - Buckets for R1: G0, G1, ... Gk
 - Buckets for R2: H0, H1, ... Hk

Algorithm

- 1) Hash R1 tuples into G buckets
- 2) Hash R2 tuples into H buckets
- 3) For i = 0 to k do
 match tuples in Gi, Hi buckets

UB CSE 562 Spring 2009

16

Simple Example

hash: even/odd

R1	R2	Buckets	
2	5	Even: 2 4 8	4 12 8 14
4	4		
3	12		
5	3		
8	13		
9	8		
	11		
	14		
		Odd: 3 5 9	5 3 13 11

UB CSE 562 Spring 2009

17

Factors that Affect Performance

- (1) Tuples of relation stored physically together?
- (2) Relations sorted by join attribute?
- (3) Indexes exist?

UB CSE 562 Spring 2009

18

Disk-Oriented Computation Model

- There are M main memory buffers
 - Each buffer has the size of a disk block
- The input relation is read one block at a time
- The cost is the number of blocks read
- The output buffers are not part of the M buffers mentioned above
 - *Pipelining* allows the output buffers of an operator to be the input of the next one
 - We do not count the cost of writing the output

UB CSE 562 Spring 2009

19

Notation

- $B(R)$ = number of blocks that R occupies
- $T(R)$ = number of tuples of R
- $V(R, [a_1, a_2, \dots, a_n])$ = number of distinct tuples in the projection of R on a_1, a_2, \dots, a_n

UB CSE 562 Spring 2009

20

One-Pass Main Memory Algorithms for Unary Operators

- Assumption: Enough memory to keep the relation
- Projection and selection:
 - Scan the input relation R and apply operator one tuple at a time
 - Incremental cost of “on the fly” operators is 0
 - Cost depends on
 - clustering of R
 - whether the blocks are consecutive
- Duplicate elimination and aggregation
 - create one entry for each group and compute the aggregated value of the group
 - it becomes hard to assume that CPU cost is negligible
 - main memory data structures are needed

UB CSE 562 Spring 2009

21

One-Pass Nested Loop Join

- Assume $B(R)$ is less than M
- Tuples of R should be stored in an efficient lookup structure
- **Exercise:** Find the cost of the algorithm below

```
for each block Br of R do
  store tuples of Br in main memory
for each block Bs of S do
  for each tuple s of Bs
    join tuples of s with matching tuples of R
```

UB CSE 562 Spring 2009

22

Generalization of Nested-Loops

```
for each chunk of  $M-1$  blocks Br of R do
  store tuples of Br in main memory
for each block Bs of S do
  for each tuple s of Bs
    join tuples of s with matching tuples of R
```

Exercise: Compute the cost of the above algorithm

UB CSE 562 Spring 2009

23

Simple Sort-Merge Join

- Assume natural join on C
- Sort R on C using the two-phase multiway merge sort
 - if not already sorted
- Sort S on C
- Merge (opposite side)
 - assume two pointers Pr, Ps to tuples on disk, initially pointing at the start
 - sets R', S' in memory
- Remarks:
 - Very low average memory requirement during merging (but no guarantee on how much is needed)
 - **Cost:**

```
while Pr!=EOF and Ps!=EOF
  if *Pr[C] == *Ps[C]
    do_cart_prod(Pr,Ps)
  else if *Pr[C] > *Ps[C]
    Ps++
  else if *Ps[C] > *Pr[C]
    Pr++

function do_cart_prod(Pr,Ps)
  val = *Pr[C]
  while *Pr[C] == val
    store tuple *Pr in set R'
  while *Ps[C] == val
    store tuple *Ps in set S'
  output R' x S' // product
```

UB CSE 562 Spring 2009

24

Efficient Sort-Merge Join

- **Idea:** Save two disk I/O's per block by combining the second pass of sorting with the "merge"
- **Step 1:**
Create sorted sublists of size M for R and S
- **Step 2:**
Bring the first block of each sublist to a buffer
– assume no more than M sublists in all
- **Step 3:**
Repeatedly find the least C value c among the first tuples of each sublist. Identify all tuples with join value c and join them.
– When a buffer has no more tuple that has not already been considered load another block into this buffer

UB CSE 562 Spring 2009

25

Efficient Sort-Merge Join Example

R	C	RA
	1	r_1
	2	r_2
	3	r_3
	⋮	
	20	r_{20}

S	C	SA
	1	s_1
	⋮	
	5	s_5
	16	s_{16}
	⋮	
	20	s_{20}

- Assume that after first phase of multiway sort we get 4 sublists, 2 for R and 2 for S
- Also assume that each block contains two tuples

R	3 7	8 10	11 13	14 16	17 18
	1 2	4 5	6 9	12 15	19 20

S	1 3	5 17	
	2 4	16 18	19 20

UB CSE 562 Spring 2009

26

Two-Pass Hash-Based Algorithms

- General Idea: Hash the tuples of the input arguments in such a way that all tuples that must be considered together will have hashed to the same hash value
– If there are M buffers pick $M-1$ as the number of hash buckets
- Example: Duplicate Elimination
– Phase 1: Hash each tuple of each input block into one of the $M-1$ bucket/buffers. When a buffer fills, save to disk
– Phase 2: For each bucket:
– load the bucket in main memory
– treat the bucket as a small relation and eliminate duplicates
– save the bucket back to disk
– **Catch:** Each bucket has to be less than M
– **Cost:**

UB CSE 562 Spring 2009

27

Hash-Join Algorithms

- Assuming natural join, use a hash function that
– is the same for both input arguments R and S
– uses only the join attributes
- **Phase 1:** Hash each tuple of R into one of the $M-1$ buckets R_i and similar each tuple of S into one of S_i
- **Phase 2:** For $i=1 \dots M-1$
load R_i and S_i in memory
join them and save result to disk
- **Question:** What is the maximum size of buckets?
- **Question:** Does hashing maintain sorting?

UB CSE 562 Spring 2009

28

Index-Based Join: Simplest Version

- Assume that we do natural join of $R(A,B)$ and $S(B,C)$ and there is an index on S

```
for each Br in R do
  for each tuple r of Br with B value b
    use index of S to find
      tuples {s1, s2, ..., sn} of S with B=b
    output {rs1, rs2, ..., rsn}
```
- **Cost:** Assuming R is clustered and non-sorted and the index on S is clustered on B then $B(R) + T(R)B(S)/V(S,B) + \text{some for reading index}$

UB CSE 562 Spring 2009

29

Opportunities in Joins Using Sorted Indexes

- Do a conventional Sort-Join avoiding the sorting of one or both of the input operands

UB CSE 562 Spring 2009

30

Example 1(a)

Iteration Join $R1 \bowtie R2$

- Relations not contiguous
- Recall $\left\{ \begin{array}{l} T(R1) = 10,000 \\ T(R2) = 5,000 \\ S(R1) = S(R2) = 1/10 \text{ block} \\ \text{MEM} = 101 \text{ blocks} \end{array} \right.$

Cost: for each $R1$ tuple:

[Read tuple + Read $R2$]

Total = 10,000 [1 + 5000] = 50,010,000 IOs

UB CSE 562 Spring 2009

31

Can we do better?

Use our memory

- (1) Read 100 blocks of $R1$
- (2) Read all of $R2$ (using 1 block) + join
- (3) Repeat until done

UB CSE 562 Spring 2009

32

Cost

- for each R1 chunk:
Read chunk: 1000 IOs
Read R2 $\frac{5000}{6000}$ IOs
- $$\text{Total} = \frac{10,000}{1,000} \times 6000 = 60,000 \text{ IOs}$$

UB CSE 562 Spring 2009

33

Can we do better?

- Reverse Join Order: $R2 \bowtie R1$**

$$\text{Total} = \frac{5000}{1000} \times (1000 + 10,000) =$$

$$5 \times 11,000 = 55,000 \text{ IOs}$$

UB CSE 562 Spring 2009

34

Example 1(b)

Iteration Join $R2 \bowtie R1$

- Relations contiguous

Cost

For each R2 chunk:

$$\begin{aligned} \text{Read chunk: } & 100 \text{ IOs} \\ \text{Read R1: } & \frac{1000}{1,100} \text{ IOs} \end{aligned}$$

$$\text{Total} = 5 \text{ chunks} \times 1,100 = 5,500 \text{ IOs}$$

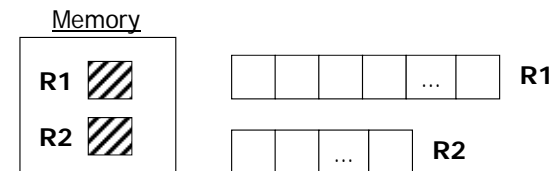
UB CSE 562 Spring 2009

35

Example 1(c)

Merge Join

- Both R1, R2 ordered by C; relations contiguous



$$\begin{aligned} \text{Total cost: } & \text{Read R1 cost} + \text{read R2 cost} \\ & = 1000 + 500 = 1,500 \text{ IOs} \end{aligned}$$

UB CSE 562 Spring 2009

36

Example 1(d)

Merge Join

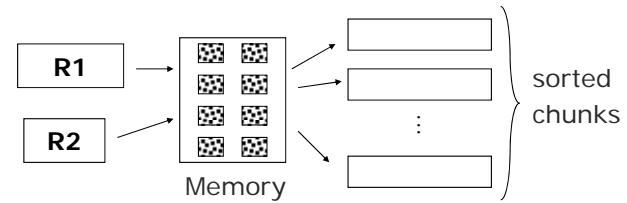
- R1, R2 not ordered, but contiguous

→ Need to sort R1, R2 first... HOW?

One Way to Sort

Merge Sort

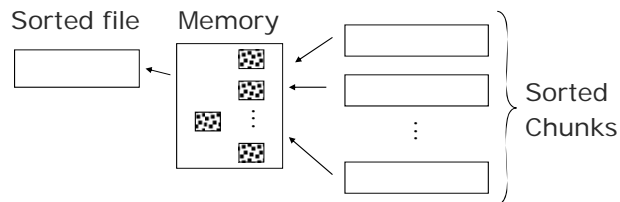
- (i) For each 100 block chunk of R:
- Read chunk
 - Sort in memory
 - Write to disk



One Way to Sort

Merge Sort

- (ii) Read all chunks + merge + write out



One Way to Sort

Cost: Sort

Each tuple is read, written,
read, written

so...

Sort cost R1: $4 \times 1,000 = 4,000$

Sort cost R2: $4 \times 500 = 2,000$

Example 1(d) (Cont.)

Merge Join

- R1,R2 contiguous, but unordered

$$\begin{aligned}\text{Total cost} &= \text{sort cost} + \text{join cost} \\ &= 6,000 + 1,500 = 7,500 \text{ IOs}\end{aligned}$$

But: Iteration cost = 5,500
so merge join does not pay off!

Example 1(d) (Cont.)

But say R1 = 10,000 blocks contiguous
R2 = 5,000 blocks not ordered

$$\begin{aligned}\text{Iterate: } \frac{5000}{100} \times (100 + 10,000) &= 50 \times 10,100 \\ &= 505,000 \text{ IOs}\end{aligned}$$

$$\text{Merge join: } 5(10,000 + 5,000) = 75,000 \text{ IOs}$$

Merge Join (with sort) WINS!

Merge Sort

How much memory do we need for merge sort?

E.g: Say I have 10 memory blocks



In General

Say k blocks in memory
x blocks for relation sort

chunks = (x/k)
size of chunk = k

chunks \leq buffers available for merge

$$\begin{aligned}\text{so... } (x/k) &\leq k \\ \text{or } k^2 &\geq x \text{ or } k \geq \sqrt{x}\end{aligned}$$

In Our Example

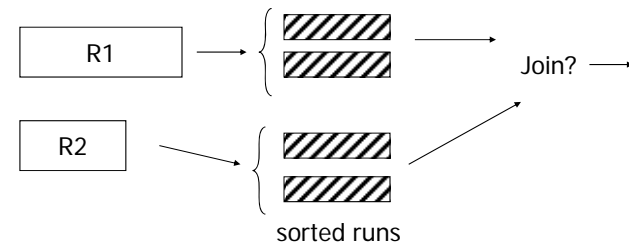
R1 is 1000 blocks, $k \geq 31.62$

R2 is 500 blocks, $k \geq 22.36$

Need at least 32 buffers

Can we improve on merge join?

Hint: do we really need the fully sorted files?



Cost of Improved Merge Join

$C = \text{Read R1} + \text{write R1 into runs}$
+ read R2 + write R2 into runs
+ join
= 2000 + 1000 + 1500 = 4500

→ Memory requirement?

Example 1(e)

Index Join

- Assume R1.C index exists; 2 levels
- Assume R2 contiguous, unordered
- Assume R1.C index fits in memory

Example 1(e) (Cont.)

Cost: Reads: 500 IOs
for each R2 tuple:
- probe index - free
- if match, read R1 tuple: 1 IO

Example 1(e) (Cont.)

What is expected # of matching tuples?

- (a) say R1.C is key, R2.C is foreign key
then expect = 1
- (b) say $V(R1.C) = 5000$, $T(R1) = 10,000$
with uniform assumption
expect = $10,000/5,000 = 2$

Total Cost with Index Join

- (a) Total cost = $500 + 5000(1)1 = 5,500$
- (b) Total cost = $500 + 5000(2)1 = 10,500$

What if index does not fit in memory?

Example: say R1.C index is 201 blocks

- Keep root + 99 leaf nodes in memory
- Expected cost of each probe is

$$E = (0) \frac{99}{200} + (1) \frac{101}{200} \approx 0.5$$

Total Cost (including probes)

$$\begin{aligned}
 &= 500 + 5000 \text{ [Probe + get records]} \\
 &= 500 + 5000 [0.5 + 2] \quad \text{uniform assumption} \\
 &= 500 + 12,500 = 13,000 \quad \text{(case b)}
 \end{aligned}$$

UB CSE 562 Spring 2009

53

So Far

not contiguous	Iterate R2 \bowtie R1	55,000 (best)
	Merge Join	_____
	Sort+ Merge Join	_____
	R1.C Index	_____
	R2.C Index	_____
contiguous	Iterate R2 \bowtie R1	5500
	Merge join	1500
	Sort+Merge Join	7500 \rightarrow 4500
	R1.C Index	5500
	R2.C Index	_____

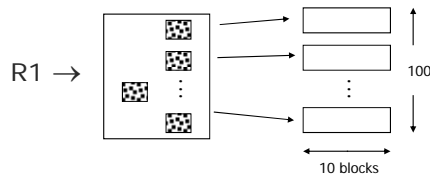
UB CSE 562 Spring 2009

54

Example 1(f)

Hash Join

- R1, R2 contiguous (un-ordered)
- Use 100 buckets
- Read R1, hash, + write buckets

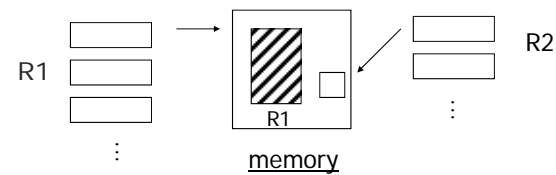


UB CSE 562 Spring 2009

55

Example 1(f) (Cond.)

- Same for R2
- Read one R1 bucket; build memory hash table
- Read corresponding R2 bucket + hash probe



- Then repeat for all buckets

UB CSE 562 Spring 2009

56

Cost

- "Bucketize:" Read R1 + write
Read R2 + write
- Join: Read R1, R2
- Total cost = $3 \times [1000+500] = 4500$

Note: this is an approximation since buckets will vary in size and we have to round up to blocks

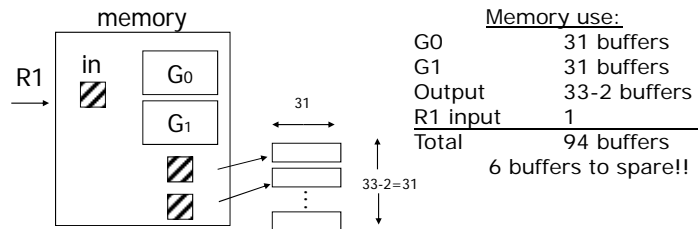
Minimum Memory Requirements

- Size of R1 bucket = (x/k)
k = number of memory buffers
x = number of R1 blocks
- So... $(x/k) < k$
- $k > \sqrt{x}$ need: $k+1$ total memory buffers

Trick

Keep Some Buckets in Memory

E.g., $k'=33$ R1 buckets = 31 blocks
keep 2 in memory



Memory use:

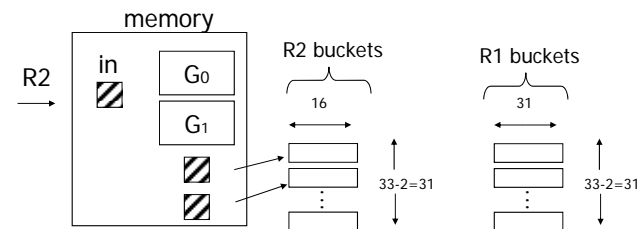
G0	31 buffers
G1	31 buffers
Output	33-2 buffers
R1 input	1
Total	94 buffers
6 buffers to spare!!	

called hybrid hash-join

Next

Bucketize R2

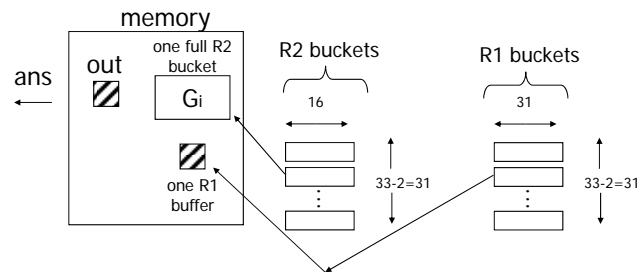
- R2 buckets = $500/33 = 16$ blocks
- Two of the R2 buckets joined immediately with G0, G1



Finally

Join remaining buckets

- for each bucket pair:
 - read one of the buckets into memory
 - join with second bucket



UB CSE 562 Spring 2009

61

Cost

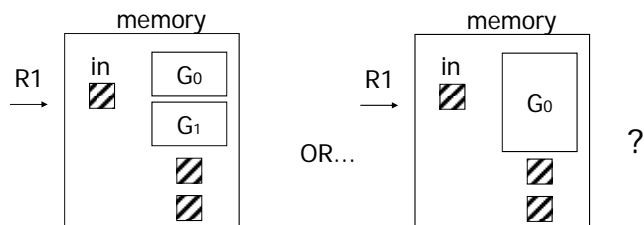
- Bucketize R1 = $1000 + 31 \times 31 = 1961$
- To bucketize R2, only write 31 buckets:
so, cost = $500 + 31 \times 16 = 996$
- To compare join (2 buckets already done)
read $31 \times 31 + 31 \times 16 = 1457$

$$\text{Total cost} = 1961 + 996 + 1457 = 4414$$

UB CSE 562 Spring 2009

62

How Many Buckets in Memory?



→ See textbook for answer...

UB CSE 562 Spring 2009

63

Another Hash Join Trick

- Only write into buckets
<val,ptr> pairs
- When we get a match in join phase,
must fetch tuples

UB CSE 562 Spring 2009

64

Another Hash Join Trick (Cont.)

- To illustrate cost computation, assume:
 - 100 <val,ptr> pairs/block
 - expected number of result tuples is 100
- Build hash table for R2 in memory
5000 tuples \rightarrow 5000/100 = 50 blocks
- Read R1 and match
- Read ~ 100 R2 tuples

<u>Total cost</u> =	Read R2:	500
	Read R1:	1000
	Get tuples:	<u>100</u>
		1600

So Far

contiguous	Iterate	5500
	Merge join	1500
	Sort+merge join	7500
	R1.C index	5500 \rightarrow 550
	R2.C index	_____
	Build R.C index	_____
	Build S.C index	_____
	Hash join	4500+
	with trick,R1 first	4414
	with trick,R2 first	_____
Hash join, pointers	1600	

Summary

- Iteration ok for "small" relations (relative to memory size)
- For equi-join, where relations not sorted and no indexes exist, hash join usually best
- Sort + merge join good for non-equi-join (e.g., R1.C > R2.C)
- If relations already sorted, use merge join
- If index exists, it could be useful (depends on expected result size)