

# Join Algorithms

## Comparing Join Algorithms

Options:

Transformations:  $R1 \bowtie R2$ ,  $R2 \bowtie R1$

- Join algorithms:
  - Iteration (nested loops join)
  - Merge join
  - Join with index
  - Hash join

Implementation of DBMS

WS 24/25  
Frankfurt UAS

Prof. Dr. Justus Klingemann

## Factors that affect performance

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

WS 24/25  
Frankfurt UAS

Prof. Dr. Justus Klingemann

## Running Example

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)

Each block holds 10 tuples →  $R1=1000$  blocks,  $R2=500$  block  
(from it: 10,000 tuples, 10 tuples per block → 1,000 blocks) ....

Implementation of DBMS

WS 24/25  
Frankfurt UAS

Prof. Dr. Justus Klingemann

## Iteration Join (Nested Loops Join)

```
for each r ∈ R1 do
    for each s ∈ R2 do
        if r.C = s.C then output r,s pair
```

## Can we do better?

### Use our memory

- (1) Read tuples from R1 into 100 main memory blocks
- (2) Read all of R2 (using 1 block) + join
- (3) Repeat until done

## Example: Iteration Join R1 $\bowtie$ R2

Relations not contiguous

Recall  $\left\{ \begin{array}{l} T(R1) = 10,000 \quad T(R2) = 5,000 \\ S(R1) = S(R2) = 1/10 \text{ block} \\ \text{MEM}=101 \text{ blocks} \end{array} \right.$

Cost: for each R1 tuple:

[Read tuple + Read R2]

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

## Example: Improved Iteration Join

If 100 blocks are used to read R1 into memory:  
number of tuples per chunk:  $100 * 10 = 1000$  tuples.....

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2:  $\frac{5000}{6000}$  IOs



$$\text{Total} = \frac{10,000}{1,000} \times 6000 = 60,000 \text{ IOs}$$

the read chunk = 1000 is derived from:  
 $10000 \text{ tuples} / 10 \text{ blocks} = 1000 \text{ tuples per chunk}....$

## Can we do better?

→ Reverse join order:  $R2 \bowtie R1$

$$\text{Total} = \frac{5000}{1000} \times (1000 + 10,000) = 5 \times 11,000 = 55,000 \text{ IOs}$$

(from earlier: 10,000 tuples, 10 tuples per block  $\rightarrow$  1,000 block

## Merge Join

### Merge join (conceptually)

```
(1) if R1 and R2 not sorted, sort them
(2) i ← 1; j ← 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  $i \leftarrow i+1$ 
```

## Modified Example:

Relations contiguous

### Cost

For each R2 chunk:

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

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

others for each R1 chunk: result 6000 I/Os ...

## Merge Join (cont.)

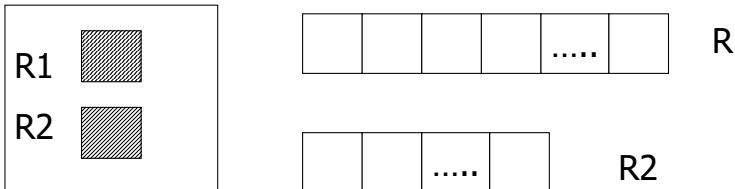
### Procedure Output-Tuples

```
While ( $R1\{i\}.C = R2\{j\}.C$ )  $\wedge$  ( $i \leq T(R1)$ ) do
     $ij \leftarrow j;$ 
    while ( $R1\{i\}.C = R2\{jj\}.C$ )  $\wedge$  ( $jj \leq T(R2)$ ) do
        [output pair  $R1\{i\}, R2\{jj\}$ ;
          $jj \leftarrow jj+1$  ]
     $i \leftarrow i+1$  ]
```

## Example: Merge Join

Assumption: Both R1, R2 ordered by C; relations contiguous

Memory



Total cost: Read R1 cost + read R2 cost

$$= 1000 + 500 = 1,500 \text{ IOs}$$

## Modified Example: Merge Join

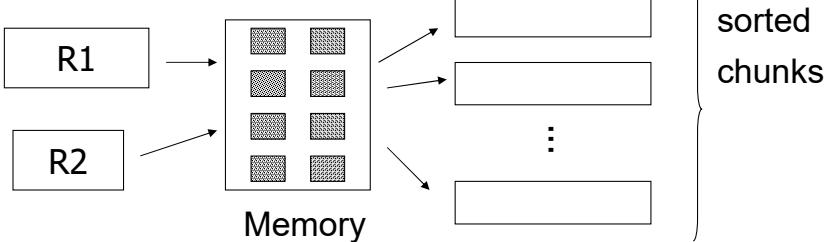
R1, R2 not ordered, but contiguous

--> Need to sort R1, R2 first.... HOW?

## Recall: 2PMMS

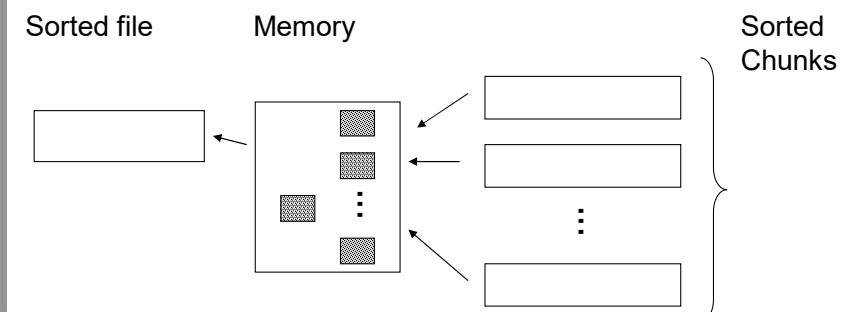
(i) For each 100 block chunk of R:

- Read chunk
- Sort in memory
- Write to disk



## 2PMMS (cont.)

(ii) Read all chunks + merge + write out



## Cost 2PMMS

Each tuple is read, written, read, written

Thus,

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

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

## Merge Join vs. Iteration Join

Iteration join is essentially quadratic whereas merge join is linear

Thus, the decision depends on the size of the relations:

Example:  $R1 = 10,000 \text{ blocks}$  both contiguous

$R2 = 5,000 \text{ blocks}$  and not ordered

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

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

Merge Join (with sort) Wins!

## Modified Example: Merge Join (cont.)

$R1, R2$  contiguous, but unordered

Total cost = sort cost + join cost

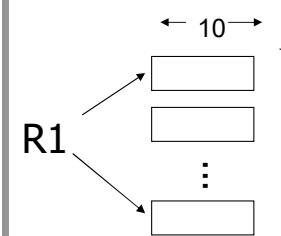
$$= 6,000 + 1,500 = 7,500 \text{ IOs}$$

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

## How much memory do we need for merge sort?

Example:

- contiguous relation with 1000 Blocks
- 10 memory blocks



100 chunks  $\Rightarrow$  to merge, need 100 input blocks!

## In general:

Say  $M$  blocks in memory

$B$  blocks for the relation to be sorted

$$\# \text{ chunks} = \lceil (B/M) \rceil \quad \text{size of chunk} = M$$

$\# \text{ chunks} < \text{buffers available for merge}$

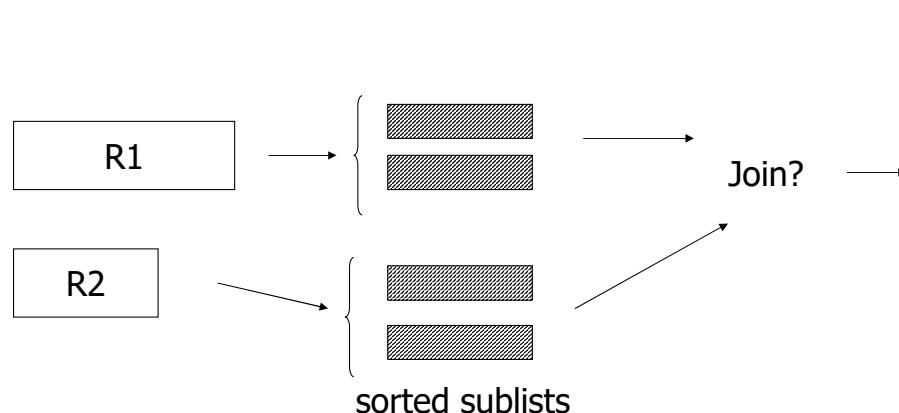
Thus  $\lceil (B/M) \rceil < M$

or approximately  $M^2 > B$  or  $M > \lceil \sqrt{B} \rceil$

## Can we improve on merge join?

Idea: do we really need the fully sorted files?

No! With enough main memory we can combine the last phase of the sorting algorithm with the actual join



## In our example

R1 is 1000 blocks,  $M > \lceil 31.62 \rceil$

R2 is 500 blocks,  $M > \lceil 22.36 \rceil$

Therefore, we need at least 33 buffer blocks in main memory

## Cost of improved merge join:

$$\begin{aligned} C = & \text{ Read R1 + write R1 into sorted sublists} \\ & + \text{ read R2 + write R2 into sorted sublists} \\ & + \text{ join} \\ & = 2000 + 1000 + 1500 = 4500 \end{aligned}$$

Limitations: more memory required during join:

- In general we require  $M^2 > B(R1) + B(R2)$
- For our example R1 and R2 we sort R1 (using 101 buffers) into 10 sorted sublists and R2 into 5 sublists.
- In the merge phase, we need at least 15 buffers, one per sublist, for input.
- That leaves additional 86 buffer blocks for records that share a C-value.

# Hash Join

## Hash join (conceptual)

- Hash function  $h$ , range  $0 \rightarrow k$
- Buckets for R1:  $G_0, G_1, \dots, G_k$
- Buckets for R2:  $H_0, H_1, \dots, H_k$

## 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  $G_i, H_i$  buckets

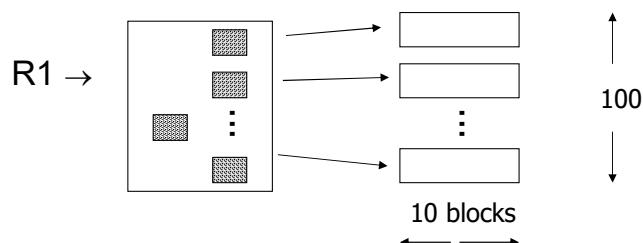
## Simple example hash: even/odd

R1	R2
2	5
4	4
3	12
5	3
8	13
9	8
	11
	14

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

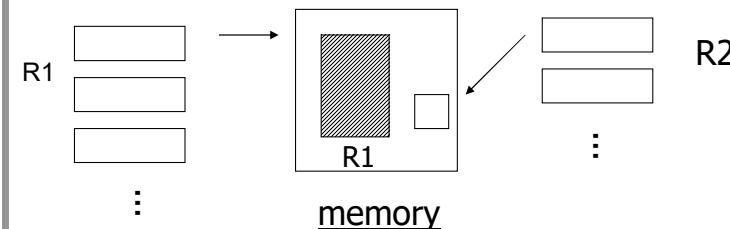
## Example Hash Join

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



## Example Hash Join

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



Then repeat for all buckets

## Cost

“Bucketize:”    Read R1 + write  
                             Read R2 + write  
 Join:                  Read R1, R2

$$\text{Total cost} = 3 \times [1000+500] = 4500$$

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

## Sort-based vs Hash-Techniques

The Hash-based algorithm has a size requirement that depends only on the smaller of the two relations rather than the sum of the argument sizes as for the optimized sort-based algorithm

The Sort-based algorithm allows us to produce a result in a sorted order and take advantage of that later

- The result might be used in another sort-based algorithm later, or
- It could be the answer to a query that is required to be produced in sorted order

The Hash-based algorithm depends on the buckets being of equal size. However, there is generally a variation in size.

Result:

- It is not possible to use buckets that on average need the whole available main-memory. We must limit them to a smaller figure
- This effect is especially prominent, if the number of different hash keys is small

## Minimum memory requirements:

Bucketizing: The number of buckets can be at most M-1  
 Assuming that all buckets have roughly the same size, the size of the bucket is  $= B / (M-1)$

M = number of memory buffers

B = number of R blocks

Joining of individual buckets: The bucket of one relation has to be completely in main memory (in detail: fit in M-1 blocks)  
 Therefore:  $B/(M-1) \leq M-1$

or approximately  $M > \lceil \sqrt{B} \rceil$

Note: In contrast to Merge join it is sufficient if this relationship holds for the smaller relation

## Different Number of Passes (1)

The execution of some algorithms relies on the availability of a certain number of memory buffers

- sort-based algorithms
- hash-based algorithms

Remind:

with

M = number of memory buffers

B = number of blocks for the relation(s)

we have to have approximately  $M^2 > B$

The algorithms use two passes

- one pass to prepare the data (sorted sublists, buckets)
- a second pass to perform the desired action (e.g., join)

## Different Number of Passes (2)

If we have relations of a larger size we can add one or more additional pass

- sort-based algorithms: the subsequences are merged to produce a smaller number of larger subsequences
- hash-based algorithms: each bucket is further divided into smaller buckets using a second hash function

### Performance

- each additional pass requires reading and writing the relation

### Size of the relation

- each additional pass increases the allowed size of the relation by a factor M
- using n passes, we have  $M^n > B$

## Different Number of Passes (3)

The other extreme: we have enough memory to accommodate one relation completely in main memory, i.e.

$$M > B$$

In this case we can omit one pass and get a trivial (one-pass) join-algorithm:

- load one relation completely in main-memory
- read the second relation one block at a time and join the tuples with the first relation in main-memory