

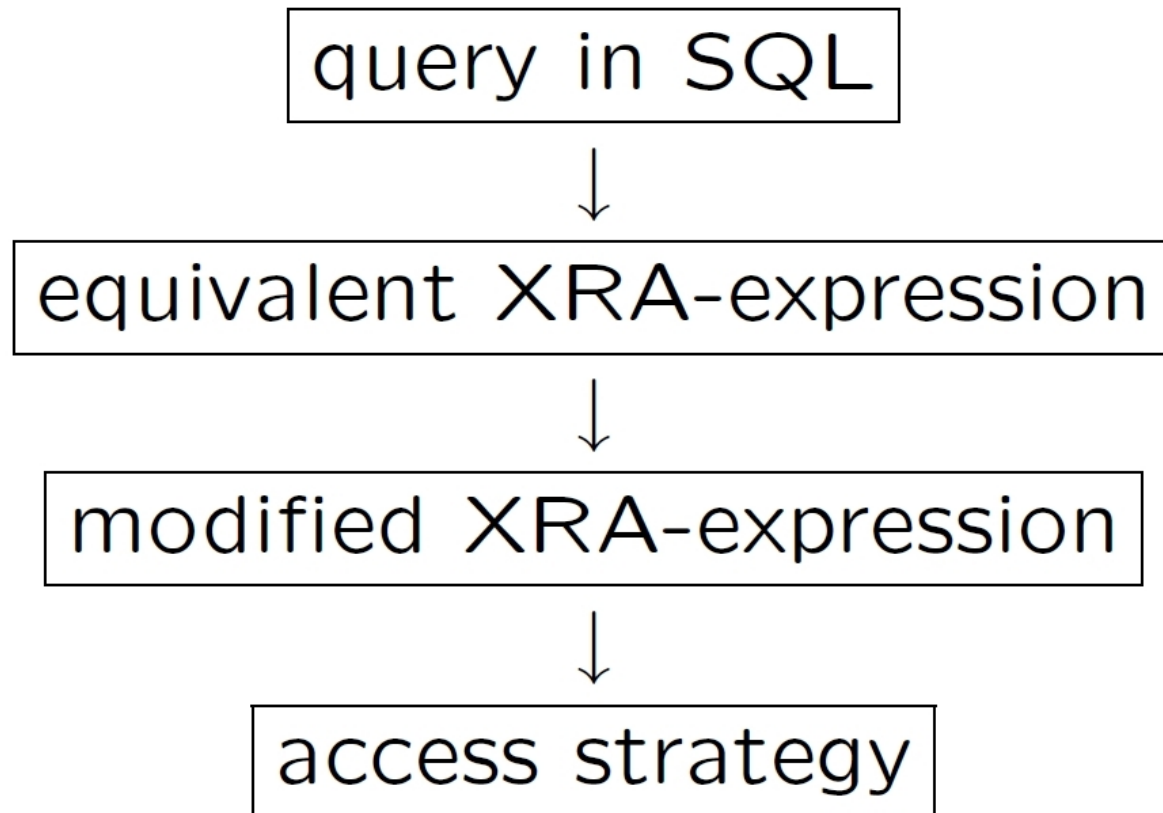
Query processing

From SQL-query to result

Let's have a look under the hood



Query processing: overview



Algebraic operators

Classical view on RA: sets

Theory of relational databases: table is a *set*

Practice (SQL): a relation is a *bag* of tuples

R

A	B
1	1
2	1
3	2
4	2

$\pi_B(R)$

B
1
2

$\pi_B(R)$

B
1
1
2
2

Bags (multisets) versus sets

Union

$$\{a,b,c\} \cup \{a,a,c,d\} = \{a,a,a,b,c,c,d\}$$

Intersection

$$\{a,a,a,b,c,c,d\} \cap \{a,a,b,e\} = \{a,a,b\}$$

Difference (minus)

$$\{a,a,a,b,c,c,d\} - \{a,a,b,e\} = \{a,c,c,d\}$$

New operators in XRA: projection

Extended projection

R

A	B
1	1
2	1
3	2

$\pi_{A, A+B}(R)$

A	AplusB
1	2
2	3
3	5

New operators in XRA: sorting

- ▶ $\tau_L(R) =$
list of tuples in R sorted on attributes in L

R

B	C
2	6
4	3
1	3

$\tau_{C,B\uparrow}(R)$

C	B
3	1
3	4
6	2

Grouping en aggregate functions

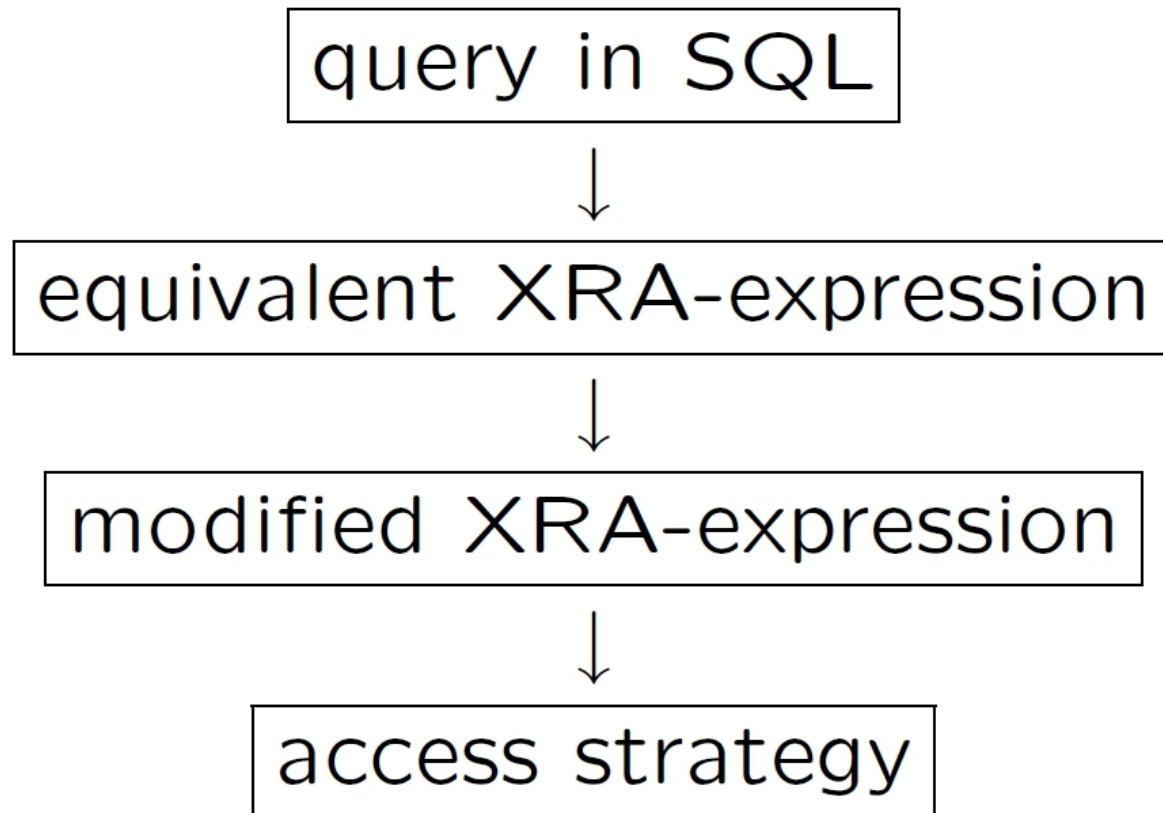
Trip

<i>Company</i>	<i>Destination</i>	<i>Price</i>
Easyjet	Barcelona	65
Ryanair	Barcelona	59
KLM	Barcelona	80
Easyjet	London	45
Lufthansa	London	58
KLM	Paris	69

$\Gamma_{destination, min(price)} (Trip)$

<i>Destination</i>	<i>MinPrice</i>
Barcelona	59
London	45
Paris	69

Query processing: overview



Recall: algebraic properties

▶ *Commutativity*

- $a + b = b + a$
- $a * b = b * a$
- $a - b \neq b - a$

▶ *Associativity*

- $(a + b) + c = a + (b + c)$
- $(a * b) * c = a * (b * c)$
- $(a - b) - c \neq a - (b - c)$

▶ *Distributivity*

- $a * (b + c) = a * b + a * c$
- $a * (b - c) = a * b - a * c$

Algebraic rewriting

cascading and commuting selections:

$$\sigma_{p \wedge q}(R) \equiv \sigma_p(\sigma_q(R)) \equiv \sigma_q(\sigma_p(R))$$

cascading projections:

$$\pi_{L1}(\pi_{L2}(R)) \equiv \pi_{L1}(R) \quad \text{IF } L1 \subseteq L2$$

commuting selections and projections:

$$\pi_L(\sigma_p(R)) \equiv \sigma_p(\pi_L(R)) \quad \text{IF } attr(p) \subseteq L$$

Algebraic rewriting

commutativity of binary operators:

$$R \bowtie_{\theta} S \equiv S \bowtie_{\theta} R \quad (?!)$$

also for \times, \cup, \cap

distribution of selection over join:

$$\sigma_{p1 \wedge p2 \wedge p3}(R \bowtie S) \equiv \sigma_{p3}(\sigma_{p1}(R) \bowtie \sigma_{p2}(S))$$

$$\text{IF } attr(p1) \subseteq attr(R), attr(p2) \subseteq attr(S)$$

distribution of selection over union:

$$\sigma_p(R \cup S) \equiv \sigma_p(R) \cup \sigma_p(S)$$

Intermezzo

- ▶ Describe how a Selection distributes over a Minus
- ▶ Describe how a Selection distributes over an Intersection
- ▶ Describe how a GroupBy distributes over a Union
- ▶ Rewrite the following expression for schema $R(ABCD), S(AEFG), T(EHK)$

$$\pi_{CG}(\sigma_{D \geq 10 \wedge E \leq 20 \wedge (G > 0 \vee K > 0)}(R \bowtie S \bowtie T))$$

Access strategies

- Focus on **selection**, **sorting** and **equijoin**
- Other operators are either simple (projection) or variants of join methods

Access strategies

- ▶ We will have a look at datastructures and algorithms to support execution of algebraic operators
- ▶ (Blok 4 INFODS) Algorithmic analysis in main memory: count the number of steps of an algorithm (= , < , +)
- ▶ Our approach: count the number of accesses to external memory (IO) and ignore data processing in CPU and main memory

How to do: duplicate elimination?

- method 1: *sorting*
external sorting of R can be done using *merge sort*
- method 2: *hashing*
hashing will be explained when discussing
join methods

IO: $4 * B$, with B = the number of blocks used to store R

External merge sort: sort R[A,B] on A

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------



MEMORY

phase 1:
2 blocks available

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

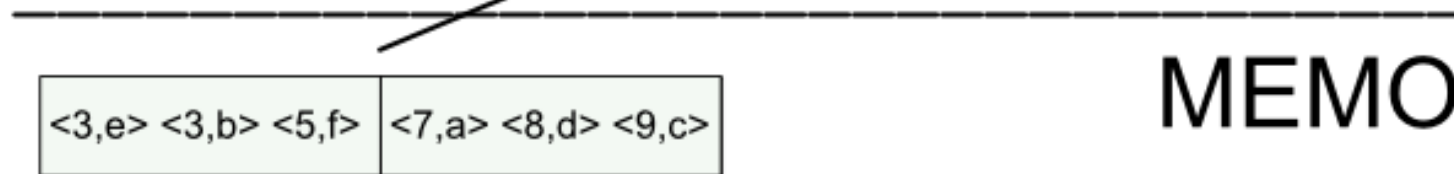
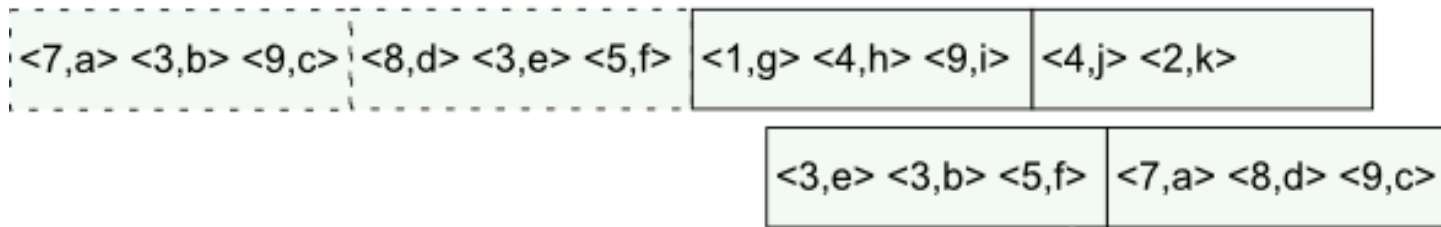


<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>
-------------------	-------------------

MEMORY

phase 1:
2 blocks available

DISK



MEMORY

phase 1:
internal sort;
first list to disk

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

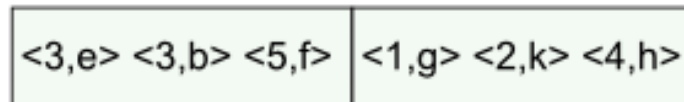
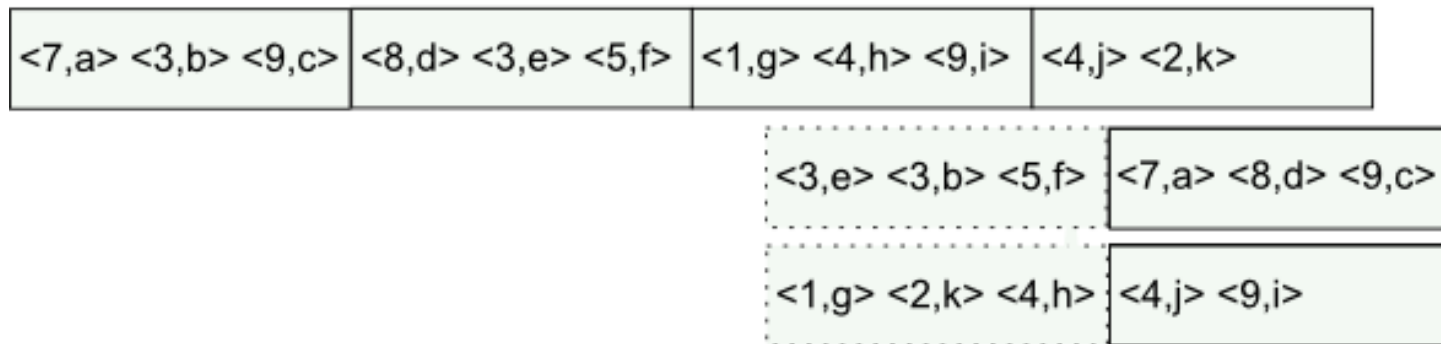
<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <4,h>	<4,j> <9,i>
-------------------	-------------

MEMORY

phase 1:
internal sort;
second list to disk

DISK



MEMORY

phase 2:
two blocks for
merging lists

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <3,e>

<3,e> <3,b> <5,f> <1,g> <2,k> <4,h>

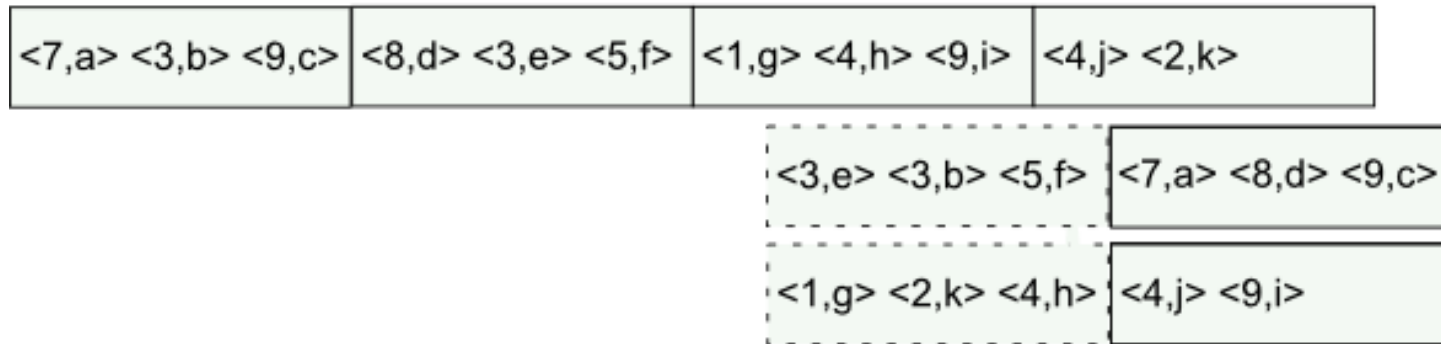
<1,g> <2,k> <3,e>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK



<1,g> <2,k> <3,e>

<3,e> <3,b> <5,f> <1,g> <2,k> <4,h>

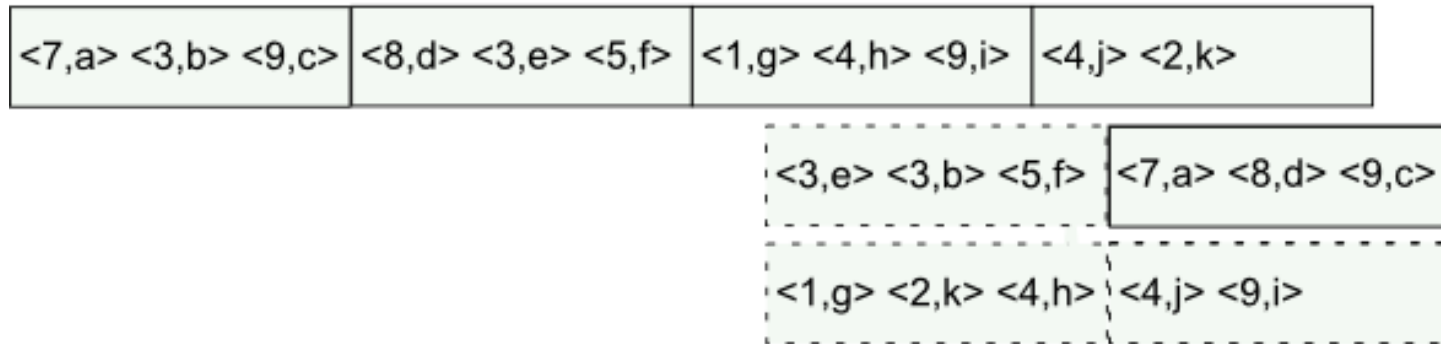
<3,b> <4,h>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK



<1,g> <2,k> <3,e>

<3,e> <3,b> <5,f> <4,j> <9,i>

<3,b> <4,h>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <3,e>	<3,b> <4,h> <4,j>
-------------------	-------------------

<3,e> <3,b> <5,f>	<4,j> <9,i>
-------------------	-------------

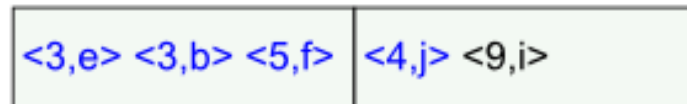
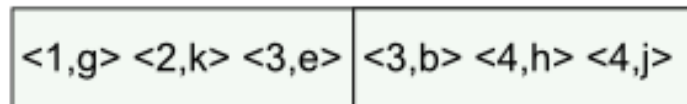
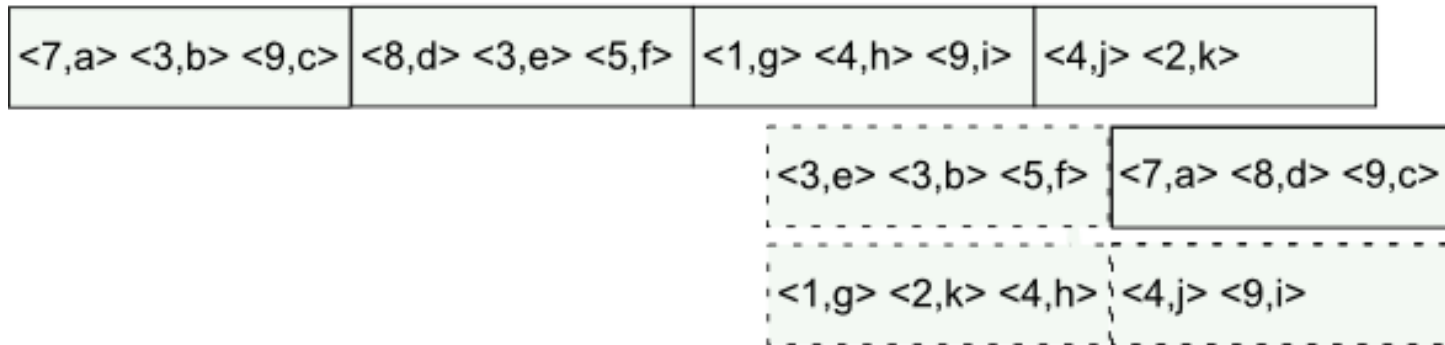
<3,b> <4,h> <4,j>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK



output window

MEMORY

phase 2:
two blocks for
merging lists

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <3,e>	<3,b> <4,h> <4,j>
-------------------	-------------------

<7,a> <8,d> <9,c>	<4,j> <9,i>
-------------------	-------------

<5,f>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <3,e>	<3,b> <4,h> <4,j>	<5,f> <7,a> <8,d>
-------------------	-------------------	-------------------

<7,a> <8,d> <9,c>	<4,j> <9,i>
-------------------	-------------

<5,f> <7,a> <8,d>

output window

MEMORY

phase 2:
two blocks for
merging lists

DISK

<7,a> <3,b> <9,c>	<8,d> <3,e> <5,f>	<1,g> <4,h> <9,i>	<4,j> <2,k>
-------------------	-------------------	-------------------	-------------

<3,e> <3,b> <5,f>	<7,a> <8,d> <9,c>
<1,g> <2,k> <4,h>	<4,j> <9,i>

<1,g> <2,k> <3,e>	<3,b> <4,h> <4,j>	<5,f> <7,a> <8,d>	<9,c> <9,i>
-------------------	-------------------	-------------------	-------------

<7,a> <8,d> <9,c>	<4,j> <9,i>
-------------------	-------------

<9,c> <9,i>

output window

MEMORY

phase 2:
two blocks for
merging lists

External sorting: analysis

- R fits in $B(R)$ blocks
- Scanning R initially requires $B(R)$ disk accesses
- Writing sorted sublists
(using window block for each bucket)
requires $B(R)$ disk accesses
- Scanning each sublist for merging requires $B(R)$ disk accesses
- Writing results again requires $B(R)$ disk accesses

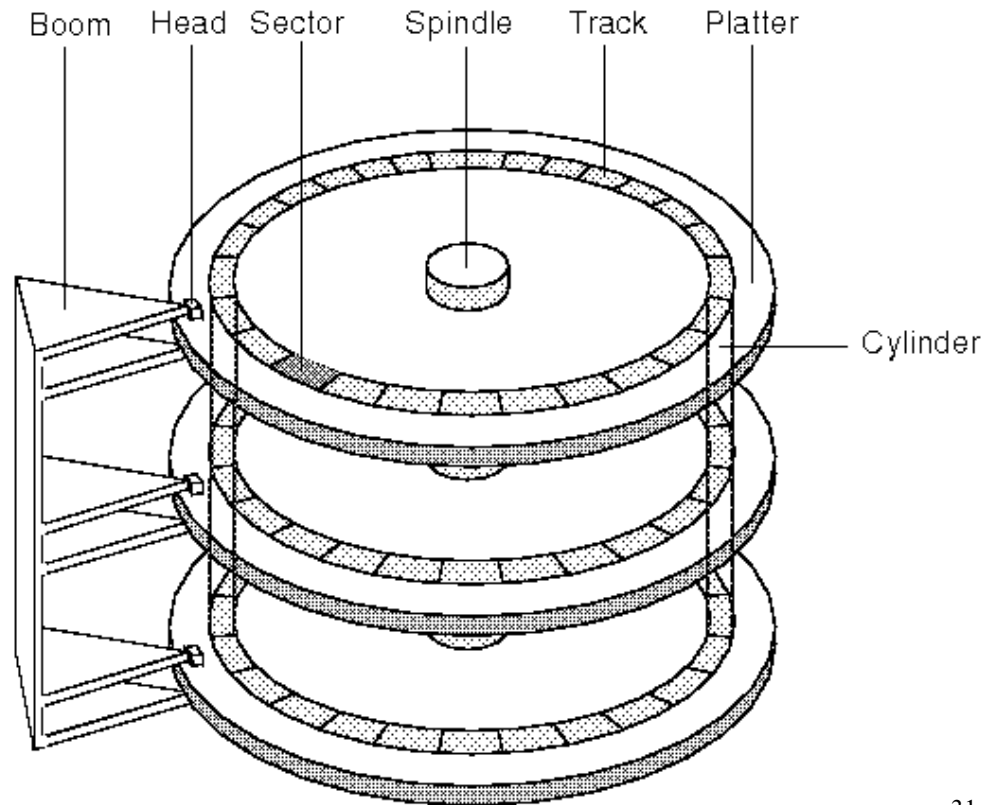
$$IO = 4 * B(R)$$

External sorting: example calculation

- $IO = 4 * B(R)$
- R contains one million tuples
- A tuple contains 400 bytes
- File containing R is 400 MB
- Disk block size is 32 kB
- Number of blocks is 12.500
- Number of IO's is 50.000
- Suppose one IO takes 5 msec
- Sorting requires 250 sec

External sorting: clustering

- Sorting requires 250 sec, but ...
- Assumption: file is *clustered*
- For instance: several blocks on one cylinder...
- ... requires 1 x disk arm positioning ...
- ... *and* less rotation latency



How to do: a selection?

$$S := \sigma_{A_1 = C_1, \dots, A_n = C_n}(R)$$

Several access paths

- Scan the complete table and check the conditions for each tuple
- If possible, use an index on an attribute contained in A_1, \dots, A_n

How to do: a selection?

$$S := \sigma_{A1 = C1, \dots, An = Cn}(R)$$

Suppose you have more than one index available

- *Option 1:* use both indices and calculate the intersection
- *Option 2:* use the most selective index

Choosing a join order

- Suppose you have to join three tables: R, S and T
- Option 1: $(R \bowtie S) \bowtie T$
- Option 2: $R \bowtie (S \bowtie T)$
- Option 3: $S \bowtie (R \bowtie T)$
- Which one is the best?
- Estimate the size of the intermediate result
- Choose the method with the smallest size
- For more than 3 relations, this is a heuristic method
- More sophisticated methods exist, but the search space grows exponentially in the number of relations

Statistics supporting estimations

$T(R)$: the number of tuples in R

$B(R)$: the number of blocks of R on disk

$V(R,A)$: the number of different values of attribute A in R

Challenge: try to estimate the effect of algebraic operators on statistics of intermediate results

Statistics supporting estimations

$$R' := \sigma_{A=c}(R)$$

$$T(R') = \dots$$

$$T(R') = T(R)/V(R,A)$$

Assumption: homogeneous distribution of A-values in R

Better: histograms

Disadvantage: maintenance

Alternative (very large databases): sampling

Statistics supporting estimations

$$U := R \bowtie_{\theta} S \qquad \theta: R.A = S.A$$

$$T(U) = \dots$$

$$T(U) = T(R) * T(S) / V(S, A)$$

$$T(U) = T(S) * T(R) / V(R, A)$$

Choose minimum?!

How to calculate ... a join?

$$U := R \bowtie_{\theta} S$$

$$\theta: R.A = S.A$$

Horrible feeling: the number of IO's required to calculate the result is proportional to

$$T(R) * T(S)$$

... or at least $B(R) * B(S)$

How to calculate ... a join?

$$T := R \bowtie_{\theta} S$$

$$\theta: R.A = S.A$$

- Block-nested loop
- Index-nested loop
- Sort-Merge
- Hash-Join

Buffer in main memory: M pages/blocks

How to calculate ... a join?

Block-nested loop

S = smallest relation (nr of blocks)

```
foreach chunk of M-1 blocks of S do  
  read these blocks into main memory;  
  foreach block B of R do {  
    read B into the free memory buffer;  
    check all possible combinations  
    of tuple t1 in chunk and t2 in B;  
    if (t1.A = t2.A) write the join of these  
      tuples to output;  
  }
```


How to calculate ... a join?

Block-nested loop

S = smallest relation (nr of blocks)

$$\text{IO: } B(S) + B(R) * \lceil B(S)/(M-1) \rceil$$

But note that if S fits in main memory and we have at least one buffer free:

$$\text{IO: } B(S) + B(R) \quad // \text{ optimal !}$$

How to calculate ... a join?

Index-nested loop

assumption: index on S.A

```
foreach block B of R do  
    foreach tuple t in B do {  
        suppose t.A = a;  
        use the index to find all t2 in S  
        with t2.A = a;  
        write the join of t with each t2 to  
        output;  
    }
```

How to calculate ... a join?

Analysis index-nested loop

c = cost of access index (~ 2 for B-tree)

μ = average number of tuples found
(estimate from statistics)

$$\text{IO: } B(R) + (c + \mu) T(R)$$

$$\mu \sim T(S)/V(S,A)$$

If A is superkey in S : $\mu = 1$

How to calculate ... a join?

Sort-merge

1. (If necessary) sort R on A
2. (If necessary) sort S on A
3. **repeat**
 - read the blocks from
R and S containing the smallest
common A-values;
 - join the tuples in these blocks;**until** R is empty or S is empty

How to calculate ... a join?

Sort-merge

A	B
a	11
b	12
a	13
c	14
c	15
a	16

A	C
b	21
a	22
c	23
b	24
c	25
a	26

1. Sort

A	B
a	11
a	13
a	16
b	12
c	14
c	15

A	C
a	22
a	26
b	21
b	24
c	23
c	25

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21

A	B	C
a	11	22
a	11	26
a	13	22
a	13	26
a	16	22
a	16	26

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
b	12	b	21
c	14	b	24
c	15	c	23

A	B	C
a	11	22
a	11	26
a	13	22
a	13	26
a	16	22
a	16	26

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
b	12	b	21
c	14	b	24
c	15	c	23

A	B	C
...
a	16	26
b	12	21
b	12	24

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
c	14	c	23
c	15	c	25

A	B	C
...
b	12	24

2. Merge

A	B	A	C
a	11	a	22
a	13	a	26
a	16	b	21
b	12	b	24
c	14	c	23
c	15	c	25

buffer (mem)

A	B	A	C
c	14	c	23
c	15	c	25

A	B	C
...
b	12	24
c	14	23
c	14	25
c	15	23
c	15	25

Sort-merge Join on A: analysis

- R and S fit in $B(R)$ and $B(S)$ blocks
- Sorting R and S requires $4 * (B(R) + B(S))$ disk accesses (if necessary)
- Scanning the sorted sublists requires (roughly) $B(R) + B(S)$ disk accesses
- Ignore writing resulting tuples in analysis (all methods have to do that)

$$IO = 5 * (B(R) + B(S))$$

Applying some smart tricks:

$$IO = 3 * (B(R) + B(S))$$

How to calculate ... a join?

Hash Join

1. Choose the size M (nr of buckets) of the hash table
2. Choose a hash function for the domain of A with codomain $0..M-1$
3. Hash every tuple of R to the corresponding bucket
4. Hash every tuple of S to the corresponding bucket
5. Get each bucket into main-memory and construct the resulting tuples

Join R[A,B] with S[A,C]

$h(a) = a \text{ DIV } 10$

A	B
12	a
3	b
29	c
7	d
13	e
12	f
27	g

A	C
29	h
7	i
8	j
28	k
12	l

Hash Join on A

A	B	A	C
12	a	29	h
3	b	7	i
29	c	8	j
7	d	28	k
13	e	12	l
12	f		
27	g		

Bucket: $h(a) = 0$

A	B	A	C
---	---	---	---

Bucket: $h(a) = 1$

A	B	A	C
---	---	---	---

Bucket: $h(a) = 2$

A	B	A	C
---	---	---	---

Hash Join on A

A	B	A	C
12	a	29	h
3	b	7	i
29	c	8	j
7	d	28	k
13	e	12	l
12	f		
27	g		

Bucket: $h(a) = 0$

A	B	A	C
3	b	7	i
7	d	8	j

Bucket: $h(a) = 1$

A	B	A	C
12	a	12	l
13	e		
12	f		

Bucket: $h(a) = 2$

A	B	A	C
29	c	29	h
27	g	28	k

Hash Join on A

Bucket:
 $h(a) = 0$

A	B	A	C
3	b	7	i
7	d	8	j

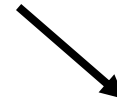
Bucket:
 $h(a) = 1$

A	B	A	C
12	a	12	l
13	e		
12	f		

Bucket:
 $h(a) = 2$

A	B	A	C
29	c	29	h
27	g	28	k

Join each bucket
in main memory:



A	B	A	C
3	b	7	i
7	d	8	j

adds:

A	B	C
7	d	i

Bucket:
 $h(a) = 0$

A	B	A	C
3	b	7	i
7	d	8	j

Bucket:
 $h(a) = 1$

A	B	A	C
12	a	12	l
13	e		
12	f		

Bucket:
 $h(a) = 2$

A	B	A	C
29	c	29	h
27	g	28	k

Join each bucket
 in main memory:

A	B	A	C
12	a	12	l
13	e		
12	f		

adds:

A	B	C
7	d	i
12	a	l
12	f	l

Bucket:
 $h(a) = 0$

A	B	A	C
3	b	7	i
7	d	8	j

Bucket:
 $h(a) = 1$

A	B	A	C
12	a	12	l
13	e		
12	f		

Bucket:
 $h(a) = 2$

A	B	A	C
29	c	29	h
27	g	28	k

Join each bucket
 in main memory:

A	B	A	C
29	c	29	h
27	g	28	k

adds:

A	B	C
7	d	i
12	a	l
12	f	l
29	c	h

Hash Join on A: analysis

- R and S fit in $B(R)$ and $B(S)$ blocks
- Scanning R and S to hash the tuples requires $B(R) + B(S)$ disk accesses
- Writing tuples to hash table (using window block for each bucket) requires roughly $B(R) + B(S)$ disk accesses
- Fetching each bucket for joining requires (roughly) $B(R) + B(S)$ disk accesses for reading
- Ignore writing resulting tuples in analysis (all methods have to do that)

$$IO = 3 * (B(R) + B(S))$$

Physical tuning

- Presence of physical structures facilitates efficient query processing
- Possibilities:
 - indexes
 - keeping tables sorted on a specific attribute
 - keeping tables clustered on a specific index
 - maintaining materialized views
- Task for the DBA: monitor performance and reconsider physical database organization
- Challenge: automatic support for optimizing physical database organization

Dual architecture: **OLTP** vs OLAP

- **OLTP** = On Line Transaction Processing
- High volumes of small updating transactions, often in combination with simple queries to identify primary key values
- Typical domain: support of commercial activities
- Real time performance requirements
- High requirements with respect to transaction integrity: concurrency, recovery, constraint satisfaction
- Term: “production database”
- Relatively low number of indexes

Dual architecture: OLTP vs OLAP

- **OLAP** = On Line Analytical Processing
- Low volumes of massive read-only queries, often changing perspective (sales per week, month, year, article, region, ...)
- Typical domain: management information systems
- No real time performance requirements, although ...
- No updates, no transaction processing; periodical reloading of preprocessed snapshots from production database
- Physical organization tuned toward read queries: many indices, materialized views, precomputed aggregates, ...
- Term: “data warehouse”

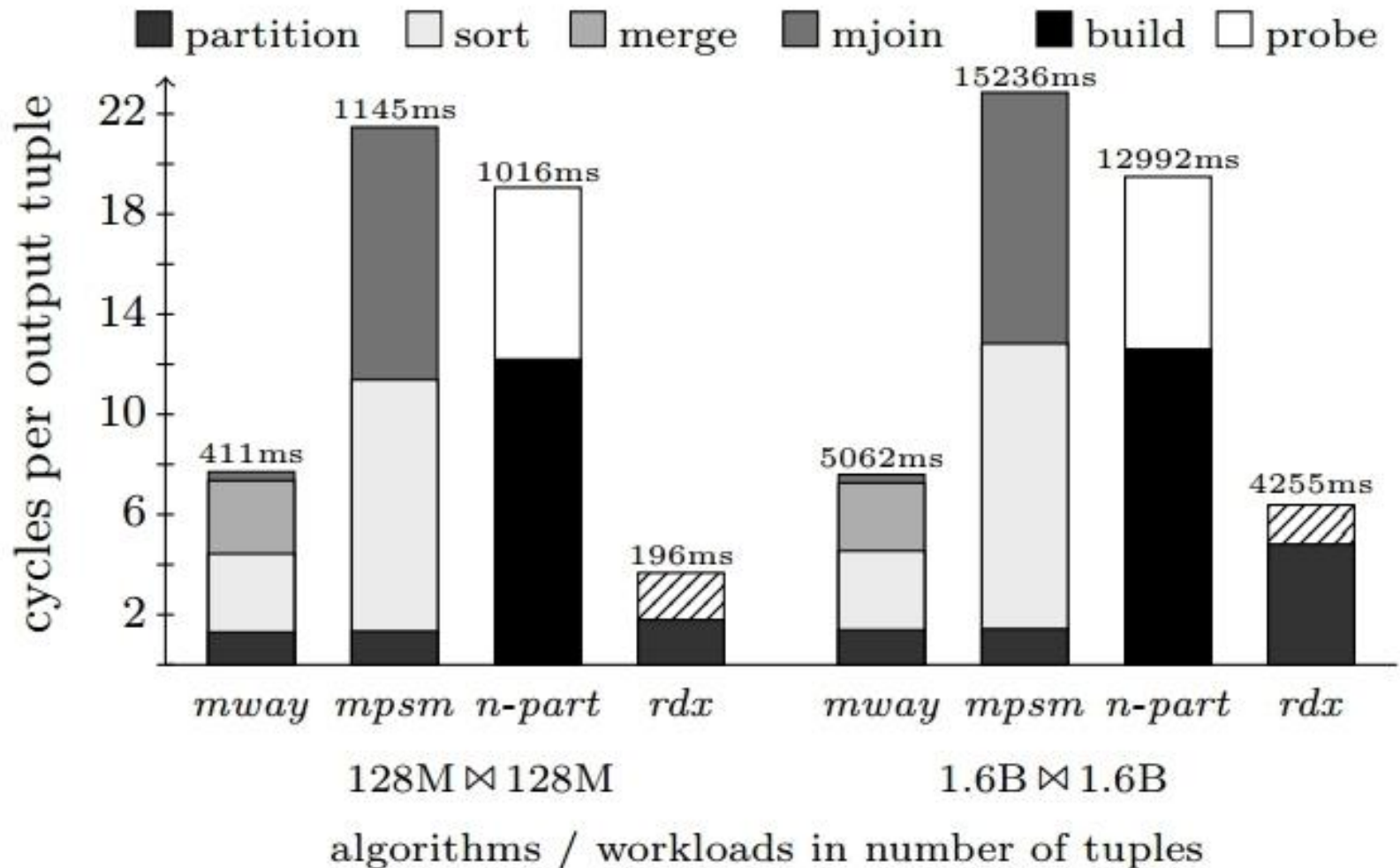
OLAP: main memory databases

- Relevant parts of database resident
- Query processing: focus on minimizing cache misses instead of minimizing IO
- Column stores instead of row stores
 - Irrelevant attributes not loaded
 - Column processing better suited for cache size
 - No overhead w.r.t. tuple management (pack/unpack)
 - But: additional joining required

OLAP: main memory databases

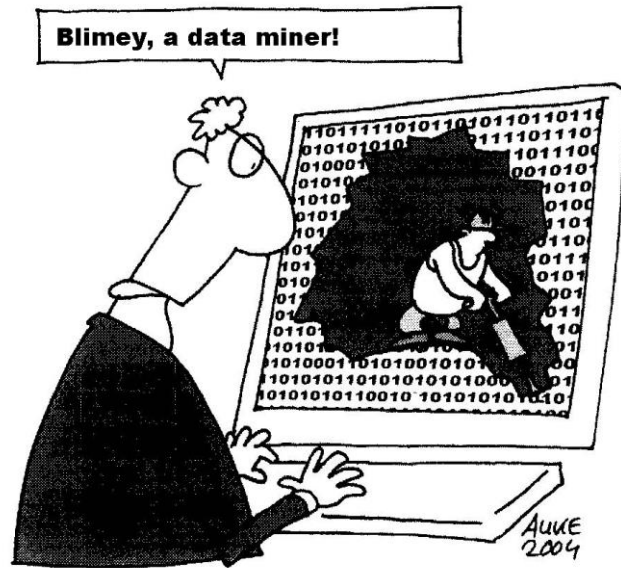
Experimental research on join processing (2014)

- Intel Sandy Bridge 4*8 cores
- 32kB L1, 256 kB L2, 20 MB L3 cache



Beyond OLAP: data mining

Discovery of interesting patterns and models in data bases



Data mining example

Market basket analysis:

Find groups of products that are often bought together



Data mining example

Market basket analysis folklore: diapers & beer

