Physical optimization (Intermediate results, Join)

Knowledge Objectives

1. Explain and exemplify the two main assumption that most DBMSs do on estimating the cardinality of query results
2. Explain the need and problem of gathering statistics
3. Enumerate five join algorithms
4. Explain the prerequisites of each join algorithm (i.e. Row Nested Loops, Block Nested Loops, Sort-Match, Hash Join)
5. Write the pseudo-code of five join algorithms

Understanding Objectives

1. Calculate the approximate size of a table, given its number of tuples, its structure, and the number of tuples and entries that fit in a disk block
2. Estimate the number of tuples in the output of a query, given the schema of the database and the tuples in the tables
3. Calculate the number of blocks necessary to store a given number of tuples, knowing the size of the attributes of each tuple and the bytes of each block
4. Identify when a join algorithm can or cannot be used in a given query or process tree
5. Given the statistics of the tables, estimate the cost of a join using each of the algorithms

Physical optimization

Consists of generating the execution plan of a query (from the best syntactic tree) considering:

- Physical structures
- Access paths
- Algorithms

Process tree

This is the tree associated to the syntactic tree that models the execution strategy

- Nodes
  - Root: Result
  - Leaves: Tables (or Indexes)
  - Internal: Intermediate tables generated by a physical operation
- Edges
  - Denote direct usage

Physical operations

- Related to relational algebra
  - Physical selection: Selection [+ projection]
  - Physical join: Join [+ projection]
  - Set operations:
    - Union [+ projection]
    - Difference [+ projection]
- Other operations:
  - Duplicate removal
  - Sorting
  - Grouping and calculating aggregates

Cost-based optimization

- The cost of the process tree is the sum of costs of each physical operation

- The cost of each operation is the sum of
  - Cost of solving it
  - Cost of writing its result

- Cost factors:
  - CPU
  - Memory access time
  - Disk access time

Phases to find the minimum cost tree

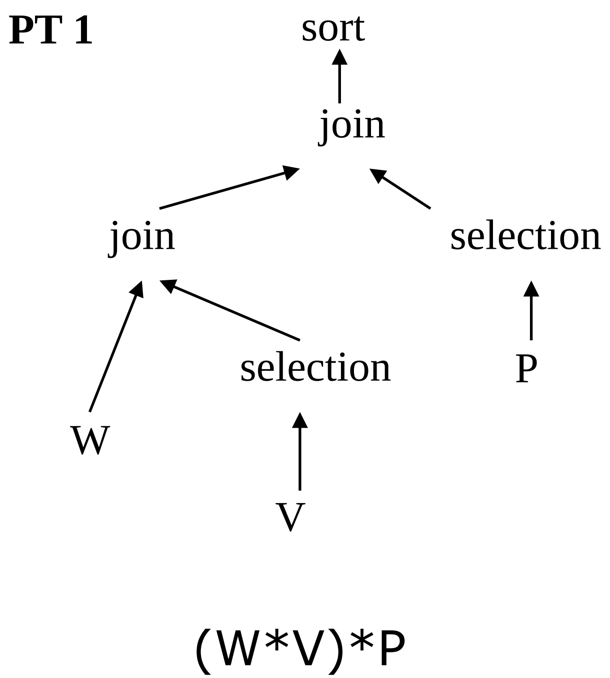
- **Phase 1**: Alternatives generation
- **Phase 2**: Intermediate results cardinality and size estimation
- **Phase 3**: Cost estimation for each algorithm and access path
- **Phase 4**: Choose the best option and generate the access plan

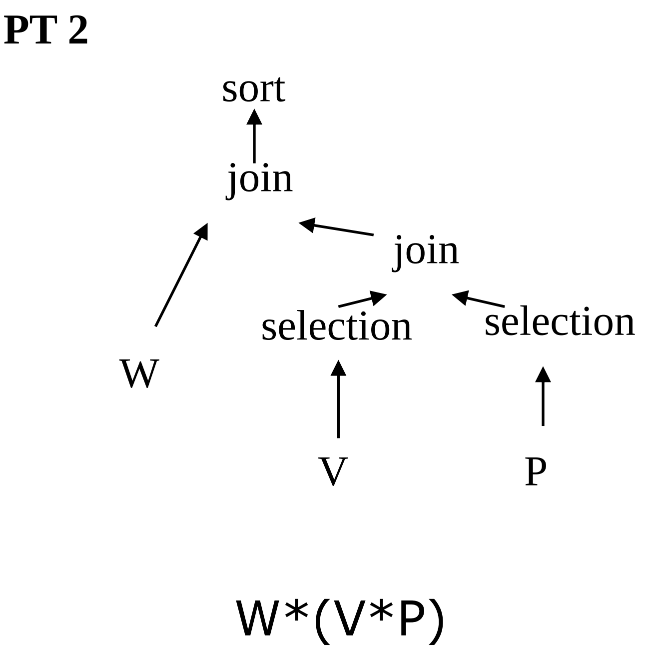
Phase 1 Execution alternatives

- Join order (combinatorial explosion)
- Access path to each tuple (depending on available structures)
- Execution algorithm for each operation
- Materialization or not of intermediate results (we will assume that they are always materialized)

Join order

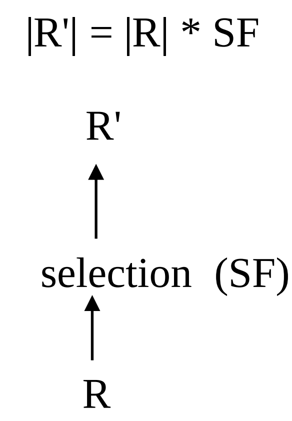
- We can generate different process trees by using the associative property of joins





Phase 2 Cardinalities estimation

- It is based on the selectivity factor ($0 \leq SF \leq 1$)
  - Next to 0 means very selective (Eg: ID)
  - Next to 1 means little selective (Eg: Sex)
- SF is only needed for selection and join
  - Estimated cardinality for selection
    |selection(R)|= SF*|R|
  - Estimated cardinality for join:
    |join(R,S)|= SF*|R|*|S|
  - Estimated cardinality for union:
    - With repetitions:
      |union(R,S)|= |R|+|S|
    - Without repetitions:
      |union(R,S)|= |R|+|S|- |join(R,S)|
  - Estimated cardinality for difference (anti-join):
    |difference(R,S)|= |R| - |join(R,S)|
- It is calculated from leaves to root
- Statistics are needed to estimate cardinalities



Statistics

- DBA is responsible for the statistics to be fresh
- Example of kinds of statistics
  - Regarding relations:
    - Cardinality
    - Number of blocks
    - Average length of records
  - Regarding attributes:
    - Length
    - Domain cardinality (maximum number of different values)
    - Number of existing different values
    - Maximum value
    - Minimum value
- Main hypothesis in most DBMS
  - Uniform distribution of values for each attribute
  - Independence of attributes

Statistics in Oracle 10g

a) ANALYZE [TABLE|INDEX|CLUSTER] <name> [COMPUTE|ESTIMATE] STATISTICS;

```
ANALYZE TABLE departments COMPUTE STATISTICS;
ANALYZE TABLE employees COMPUTE STATISTICS;
```

b) DBMS_STATS.GATHER_TABLE_STATS(<esquema>, <table>);

```
DBMS_STATS.GATHER_TABLE_STATS("username","departments");
DBMS_STATS.GATHER_TABLE_STATS("username","employees");
```

Selectivity factor of a Selection (I)

- General rule: favorable cases / possible cases
- Assuming equi-probability of values
  - SF(A=c) = 1/ndist(A)
- Assuming uniform distribution and A∈[min,max]
  - SF(A>c) = (max-c)/(max-min) (Not for integers)
    - SF(A>c) = 0 (if c≥max)
    - SF(A>c) = 1 (if c<min)
  - SF(A>v) = ½
  - SF(A<c) = (c-min)/(max-min) (Not for integers)
    - SF(A<c) = 1 (if c>max)
    - SF(A<c) = 0 (if c≤min)
  - SF(A<v) = ½
- Assuming ndist(A) big enough
  - SF(A≤x) = SF(A<x)
  - SF(A≥x) = SF(A>x)

Selectivity factor of a Selection (II)

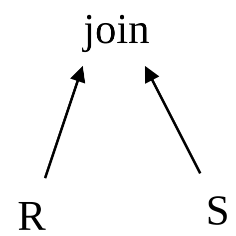
- Assuming P and Q statistically independent
  - SF(P AND Q) = SF(P)·SF(Q)
  - SF(P OR Q) = SF(P)+SF(Q)-SF(P)·SF(Q)
- SF(NOT P) = 1-SF(P)
- SF(A IN (c_1,c_2,...,c_n)) = min(1, n/ndist(A))
- SF(A BETWEEN c_1 AND c_2) = (min(c_2,max) – max(c_1,min))/(max-min) (Not for integers)
- SF(A BETWEEN v_1 AND v_2) = ¼
- SF(A BETWEEN c_1 AND v_2) = ½SF(A>c_1)
- SF(A BETWEEN v_1 AND c_2) = ½SF(A<c_2)

Selectivity factor of a Join (I)

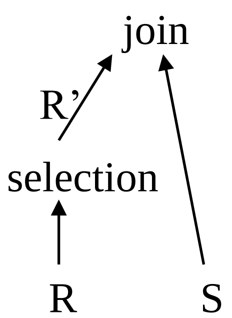
- It is difficult to approximate the general case R[AθB]S
- Depending on the comparison operator:
  - SF(R[AxB]S) = 1 ; SF(R[A<>B]S)= 1
  - SF(R[A=B]S) = 1/max(ndist(A),ndist(B))
    - There is no FK
    - One domain is subset of the other one
  - SF(R[A<B]S) = ½

Selectivity factor of a Join (II)

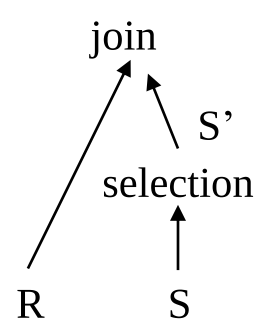
- SF(R[A=B]S) = 1/|R|
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



- SF(R'[A=B]S) = (1/|R'|) * sfr
  = 1/|R|
  - R' = selection(R) with SF = sfr
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



- SF(R[A=B]S') = (1/|R|)
  - S' = selection(S) with SF = sfr
  - S.B is FK to R.A
  - S.B is not null
  - R.A is PK



Intermediate results estimation

- Besides the cost of executing each operation, we also need to know the cost of writing intermediate results into a temporal file
  - Record length
    - $\sum$ attribute length$_i$ (+ control information)
  - Number of records per block
    - $R_R = \lfloor$block size/record length$\rfloor$
  - Number of blocks per table
    - $B_R = \lceil |R|/R_R \rceil$

Phase 3

- Join
- Selection
- Sort
- Projection

Join algorithms

- Scan (Clustered structure)
- Row Nested Loops
- Block Nested Loops
- Sort-Match
- Hash Join

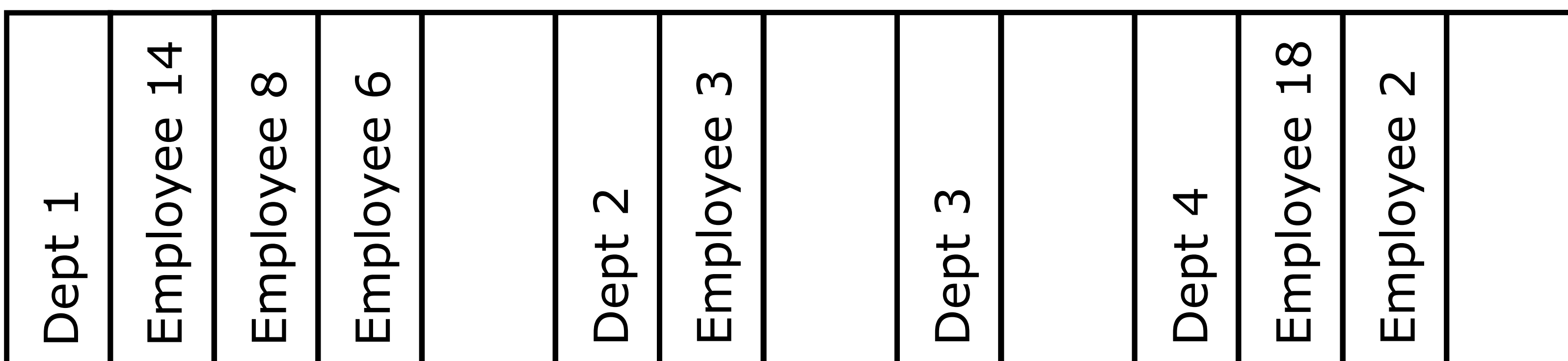
Clustered Structure

- Space
  - $\lceil 1.5B_{RS} \rceil$

$B_{RS} = B_R + B_S$

$R_{RS} = (|R|+|S|)/(B_R+B_S)$

- Access paths
  - Cost of table scan of R
    - $\lceil 1.5B_{RS} \rceil$
  - Cost of table scan of S
    - $\lceil 1.5B_{RS} \rceil$



Row Nested Loops

- Algorithm

```
for each block of R
    read block of R into a memory page
    for each tuple t in the read page
        if there is an index in S for attribute A
            go through the index of S.A using the value of t.A
            if there is any tuple satisfying the join condition
                if we are interested in attributes of S
                    go to the corresponding tuples of S
                endIf
                generate result
            endIf
        else scan the whole table S and generate result
        endIf
    endForEach
endForEach
```

u = %load·2d = (2/3)·2d
$h = \lceil \log_u |T| \rceil - 1$
k = average appearance of each value of R.A in S.A

- Cost, if we do NOT look for attributes of S (semi-join)
  - B+: $B_R + |R| \cdot (h_S + (k-1)/u_S)$
  - Hash: $B_R + |R|$
- Cost, if we DO look for attributes of S
  - B+: $B_R + |R| \cdot (h_S + (k-1)/u_S + k)$
  - Clustered: $B_R + |R| \cdot (h_S + 1 + 1.5(k-1)/R_S)$
  - Hash: $B_R + |R| \cdot (1 + k)$
- Considerations
  - It is only useful if there is an index over the join attribute
  - A hash index can only be used for equi-join
  - We cannot use this algorithm if we performed previous operation over the table
  - B_R is the real number of blocks (taking into account possible cluster)

Block Nested Loops (I)

- Algorithm

```
repeat
    read M blocks of R
    for each block of S
        read block of S into a memory page
        for each tuple t in the pages of R
            for each tuple s in the page of S
            if (t.A θ s.B) then generate result
            endIf
            endForEach
        endForEach
    endForEach
until no more blocks to read from R
```

- Cost (with M+2 memory pages)
  - $B_R + B_S \cdot \lceil B_R/M \rceil$
- Considerations
  - It can always be used (even for θ-join)
  - It is of special interest if $B_R \leq M$
  - It is not symmetric
    - It is better if the smaller table is in the external loop
    - B_R and B_S are the real number of blocks (taking into account possible cluster)

Block Nested Loops (II)

M=2

Accesses

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

E	E	I	O

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_1	R_2	S_1	

R
R
R

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_1	R_2		

R
R
R

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_1	R_2	S_2	

R
R
R
R

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E E I O

R
R
R
R

E E I O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_3	R_4	S_1	

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R
R
R
R
R
R
R

E	E	I	O

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_3	R_4		

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R
R
R
R
R
R
R

E	E	I	O

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O
R_3	R_4	S_2	

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R
R
R
R
R
R
R
R

E	E	I	O

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E E I O

R
R
R
R
R
R
R
R

E E I O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R
R
R
R
R
R
R
R

R
R
R

E	E	I	O
R_1	R_2	S_1	

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O

$R_1\ R_2\ R_3\ R_4$

$S_1\ S_2$

R	R
R	R
R	R
R	R
R	
R	
R	
R	

E	E	I	O
R_1	R_2	S_2	

$R_1\ R_2$

$S_1\ S_2\ S_3\ S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R R R R R R R R

R R R R R

E	E	I	O
R_1	R_2	S_3	

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

Accesses

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

R	R
R	R
R	R
R	R
R	R
R	R
R	
R	

E	E	I	O
R_1	R_2	S_4	

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Block Nested Loops (II)

M=2

E	E	I	O

$R_1 R_2 R_3 R_4$

$S_1 S_2$

Accesses

R	R
R	R
R	R
R	R
R	R
R	R
R	
R	

E	E	I	O

$R_1 R_2$

$S_1 S_2 S_3 S_4$

Sort-Match (I)

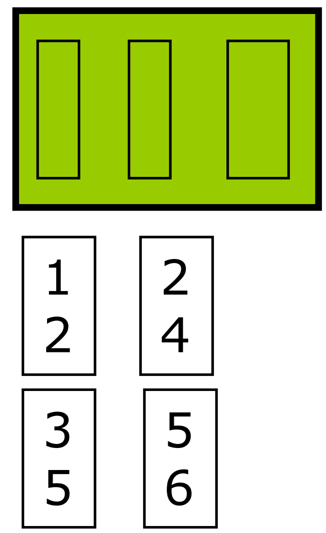
- **Algorithm** (for unique join attributes)

```
Sort R and S (if necessary)
tR:=first(R); tS:=first(S);
while not (end(R) or end(S))
   if (tR[A]<tS[A]) tR:=next(R);
   elsIf (tR[A]>tS[A]) tS:=next(S);
   else generate result from tR and tS; tR:=next(R); tS:=next(S);
   endIf
endWhile
```

- **Cost**, with M+1 memory pages
  - Sorting R (same for S):
    - If sorted: 0
    - Elsif $B_R \leq M$: $2B_R$
    - Else: $2B_R \cdot \lceil \log_M B_R \rceil$
  - Merging R and S: B_R+B_S
- **Considerations**
  - It is only useful for equi-join, <-join, >-join and anti-join
    - The given cost corresponds to equi-join and anti-join
  - It is of special interest when at least one table is Clustered
    - If both tables are sorted, we only need 3 memory pages
  - The result is already sorted
  - B_R and B_S are the real number of blocks (taking into account possible cluster). Beware: after sorting, the auxiliary tables have no empty space.

Sort-Match (II)

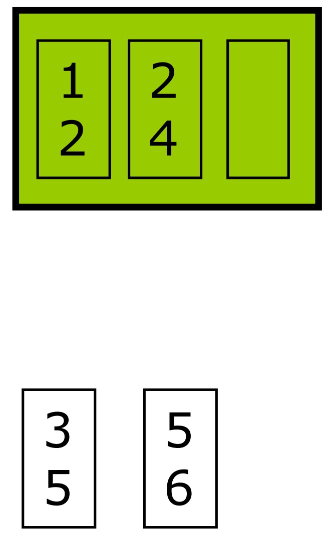
Accesses



Sort-Match (II)

Accesses

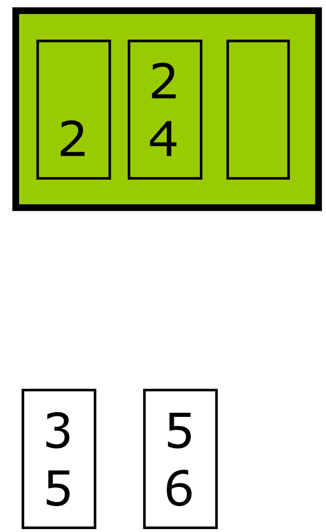
R
R



Sort-Match (II)

Accesses

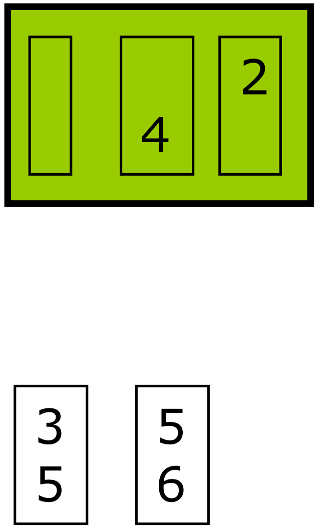
R
R



Sort-Match (II)

Accesses

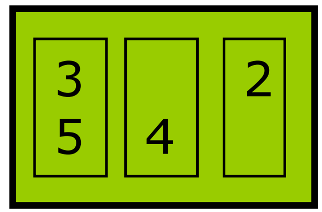
R
R

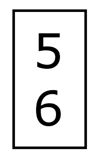


Sort-Match (II)

Accesses

R
R
R

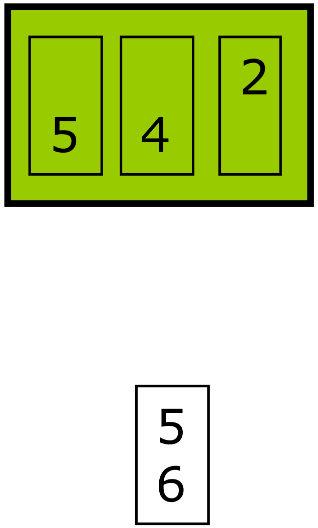




Sort-Match (II)

Accesses

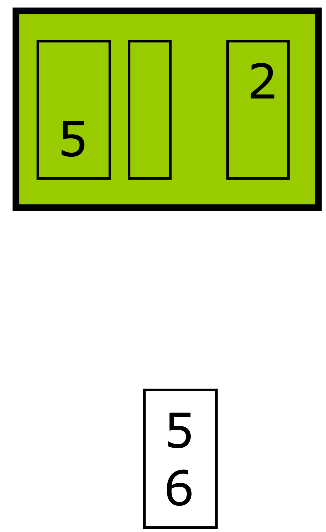
R
R
R



Sort-Match (II)

Accesses

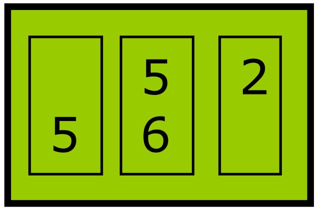
R
R
R



Sort-Match (II)

Accesses

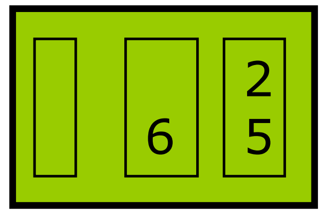
R
R
R
R



Sort-Match (II)

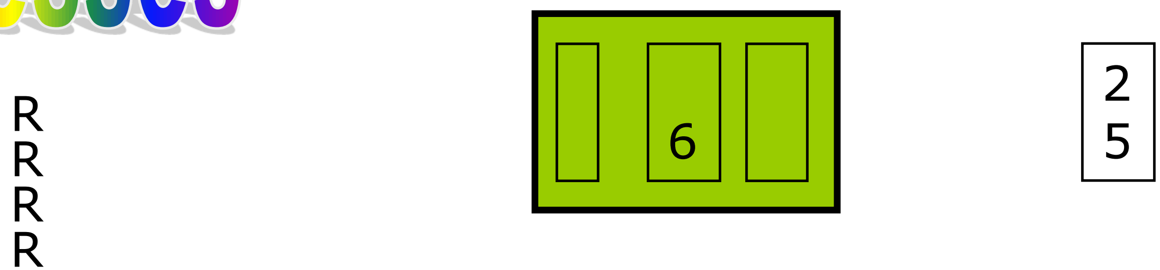
Accesses

R
R
R
R



Sort-Match (II)

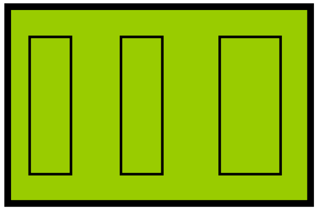
Accesses



Sort-Match (II)

Accesses

R
R
R
R



2
5

Hash Join (I)

- **Algorithm**

  Partition R into *p* parts and redistribute the tuples using a hash function
  Partition S into *p* parts and redistribute the tuples using the same hash function
  (partitions must assure that each part of one relation fits into M memory pages)
  Use Block Nested Loops *p* times (part by part)

- **Considerations**
  - It can only be used for equi-join
  - We will take $p=\lceil B_{Smaller}/M \rceil$. The size of each part of $B_{Smaller}$ is M (assuming hash results in uniform didtribution of values). Instead, the size of the other is bigger.
  - If $B_{Smaller} \leq M$, the algorithm coincides with Block Nested Loops ($p = 1$)

- **Cost**, with M+2 memory pages
  - If $B_{Smaller} \leq M$: B_R+B_S
  - If $M<B_{Smaller} \leq M^2+M$: $2B_R + 2B_S + B_R + B_S$ ($p \leq (M^2+M)/M = M + 1$)
  - If $B_{Smaller}>M^2+M$:
  - B_R and B_S are the real number of blocks (taking into account possible cluster). Beware: after redistribution, the tables have no empty space.

Hash Join (II)

Accesses

$M=1$
$B_1=B_2=2$

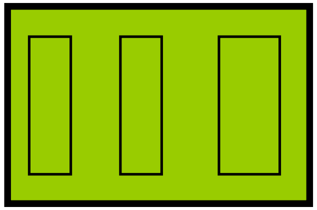


Table 1

1 2
3 5

Table 2

2 4
5 6

Hash Join (II)

Accesses

M=1
B_1=B_2=2
p=2

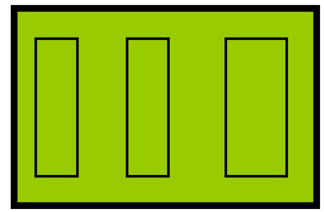


Table 1

1 2
3 5

Table 2

2 4
5 6

Hash Join (II)

Accesses

M=1
$B_1=B_2=2$
p=2

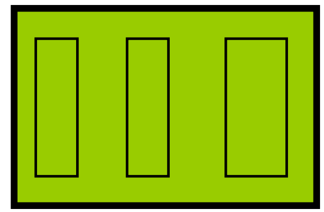


Table 1

O E

1 2
3 5

Table 2

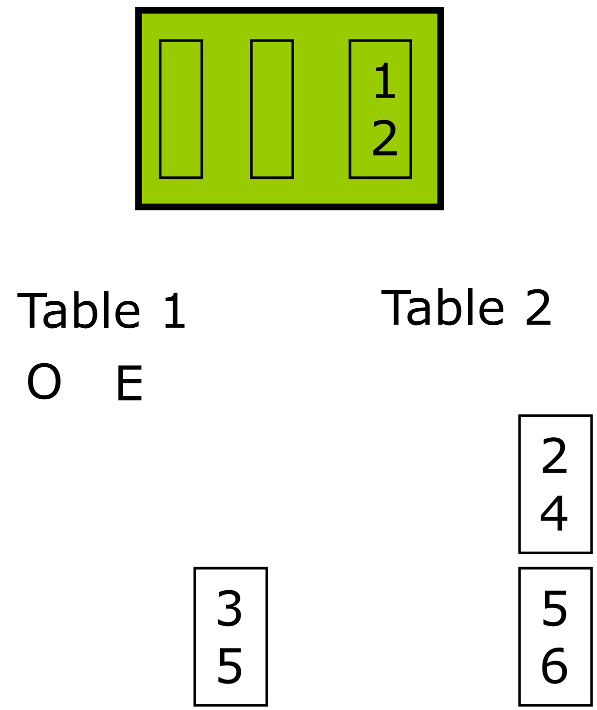
2 4
5 6

Hash Join (II)

Accesses

R

$M=1$
$B_1=B_2=2$
$p=2$

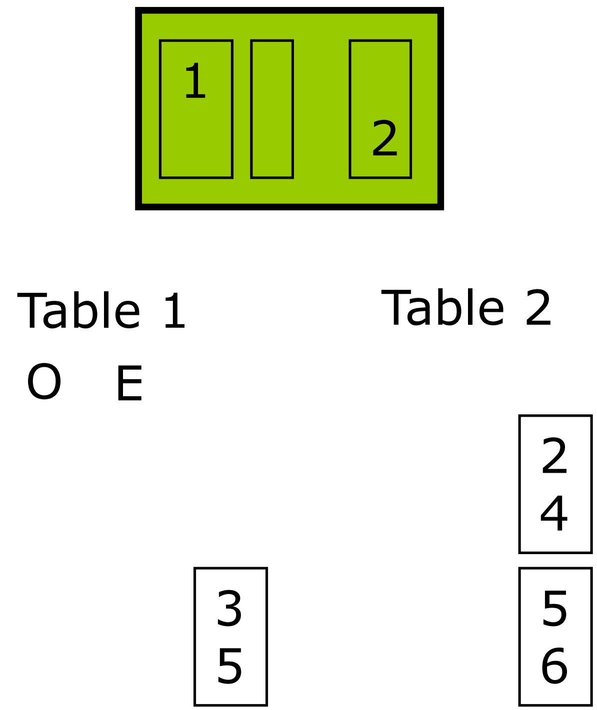


Hash Join (II)

Accesses

R

$M=1$
$B_1=B_2=2$
$p=2$



Hash Join (II)

Accesses

R

M=1
$B_1=B_2=2$
p=2

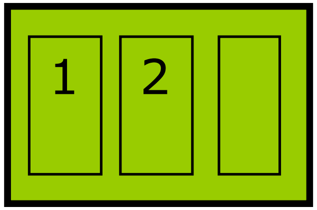


Table 1

O E

3
5

Table 2

2
4

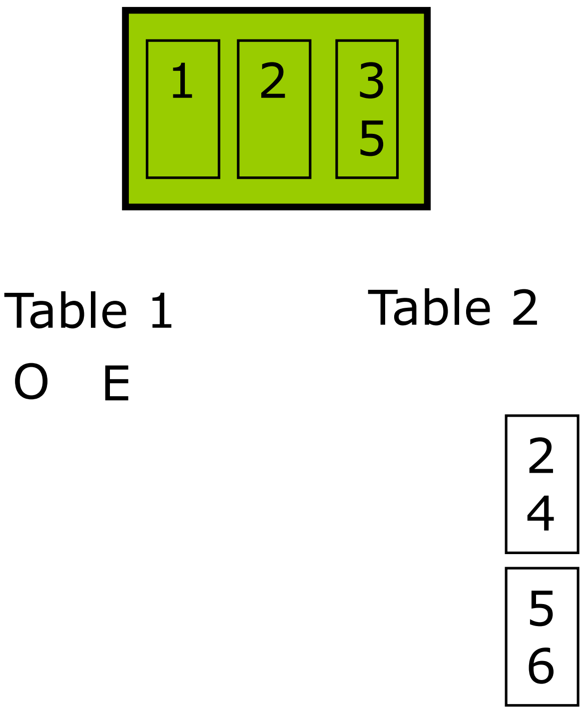
5
6

Hash Join (II)

Accesses

R
R

$M=1$
$B_1=B_2=2$
$p=2$



Hash Join (II)

Accesses

R
R

M=1
$B_1=B_2=2$
p=2

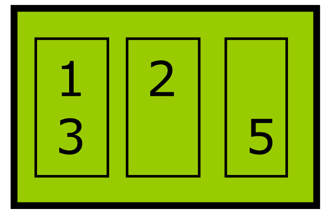
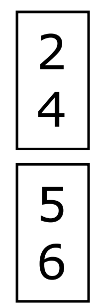


Table 1

O E

Table 2



Hash Join (II)

Accesses

R
R
W

M=1
$B_1=B_2=2$
p=2

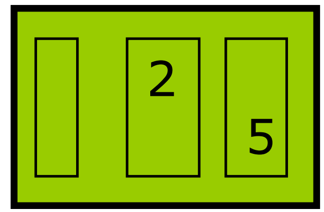


Table 1

O E

1
3

Table 2

2
4

5
6

Hash Join (II)

Accesses

R
R
W

M=1
$B_1=B_2=2$
p=2

5	2	

Table 1

O	E
1 3	

Table 2

2 4
5 6

Hash Join (II)

Accesses

R
R
W
W
W

M=1
$B_1=B_2=2$
p=2

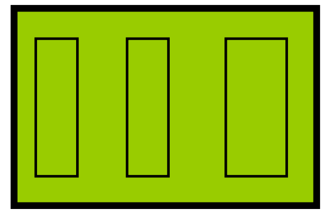


Table 1

O E

O	E
1 3	2
5	

Table 2

O E

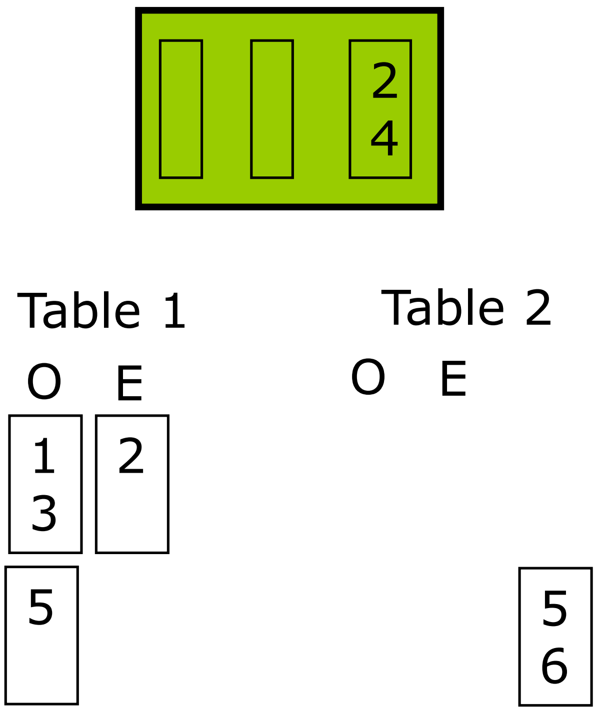
O	E
	2 4
	5 6

Hash Join (II)

Accesses

R
R
W
W
W
R

$M=1$
$B_1=B_2=2$
$p=2$

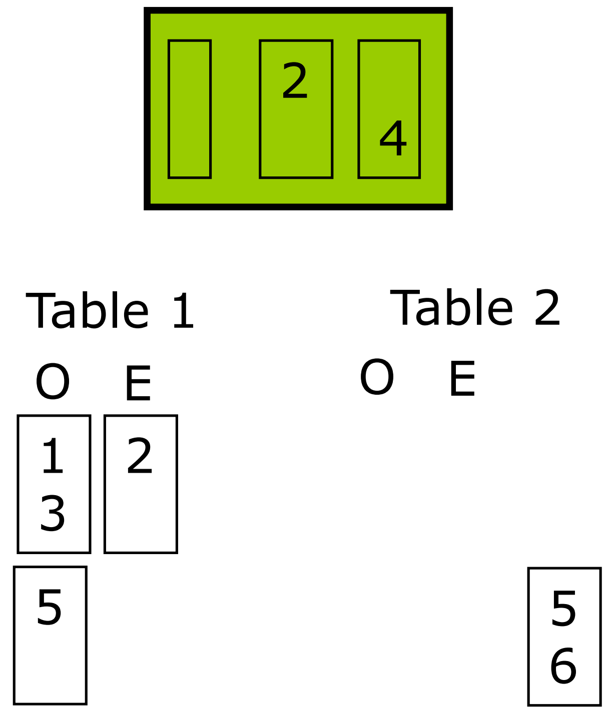


Hash Join (II)

Accesses

R
R
W
W
W
R

$M=1$
$B_1=B_2=2$
$p=2$

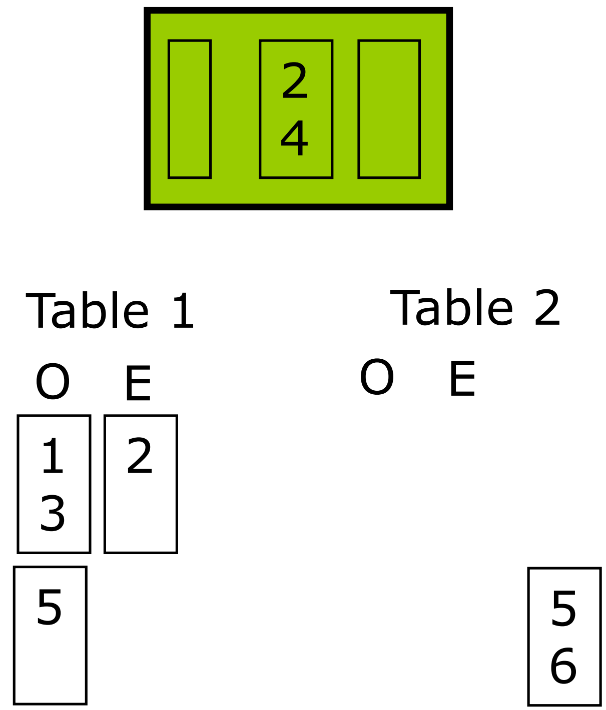


Hash Join (II)

Accesses

R
R
W
W
W
R

M=1
$B_1=B_2=2$
p=2



Hash Join (II)

Accesses

R
R
W
W
W
R
W

$M=1$
$B_1=B_2=2$
$p=2$

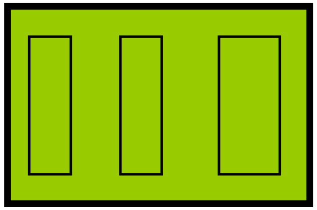


Table 1

O	E
1 3	2
5	

Table 2

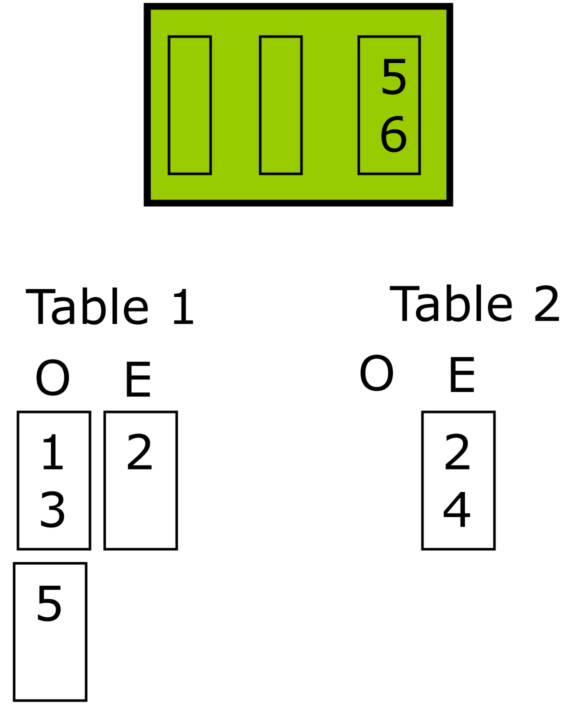
O	E
	2 4
5 6	

Hash Join (II)

Accesses

R
R
W
W
W
R
W
R

$M=1$
$B_1=B_2=2$
$p=2$

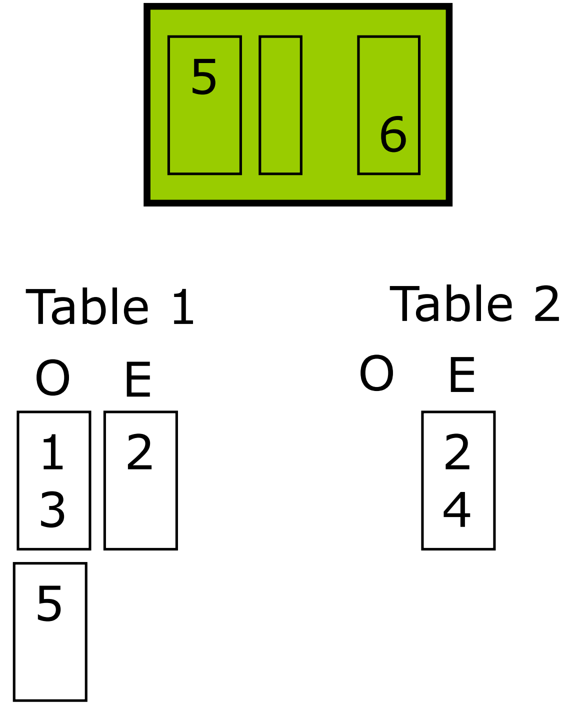


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R

M=1
$B_1=B_2=2$
p=2

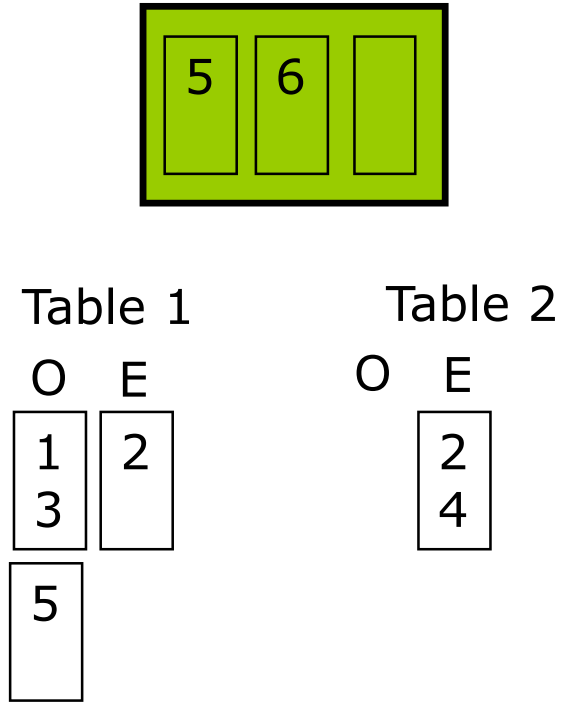


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R

$M=1$
$B_1=B_2=2$
$p=2$

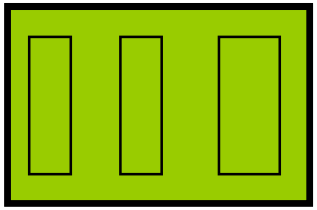


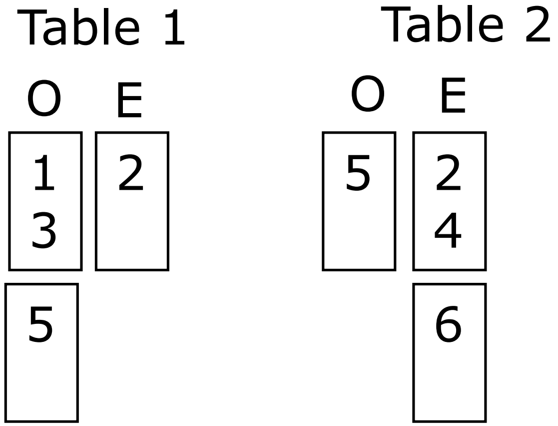
Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W

M=1
$B_1=B_2=2$
p=2



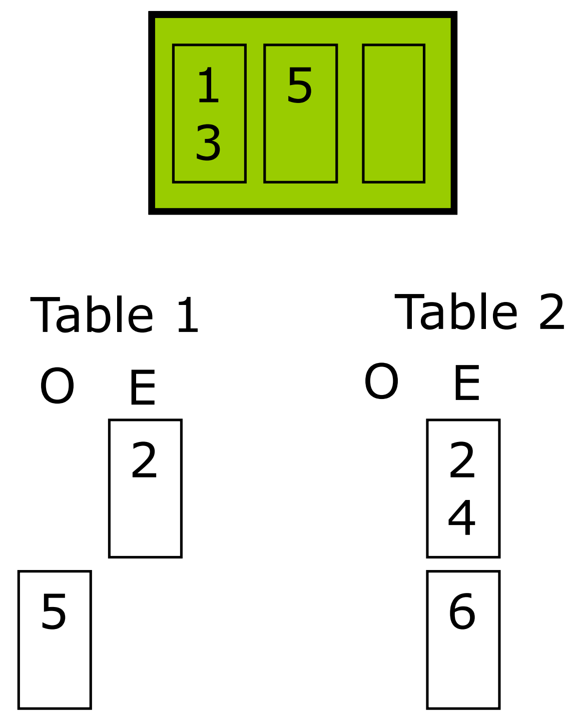


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R

M=1
$B_1=B_2=2$
p=2

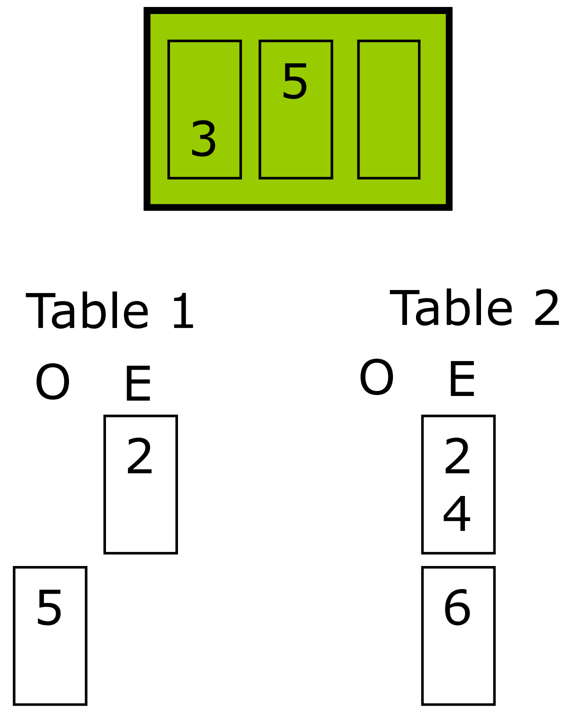


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R

M=1
$B_1=B_2=2$
p=2

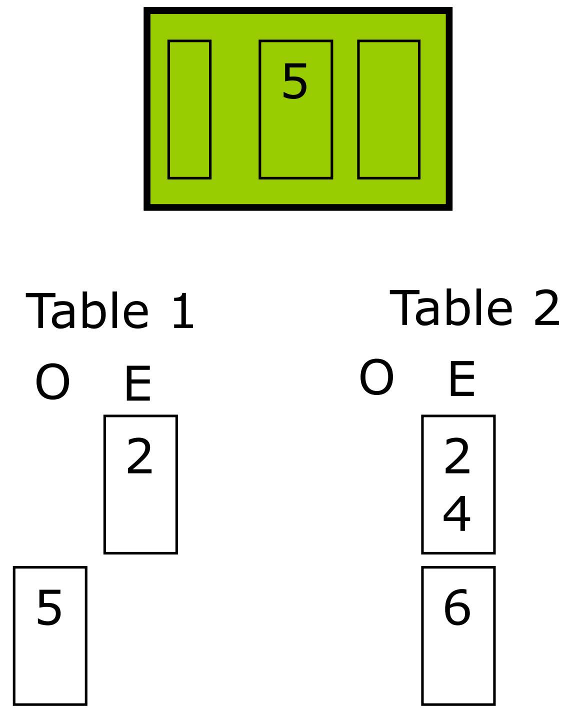


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R

$M=1$
$B_1=B_2=2$
$p=2$

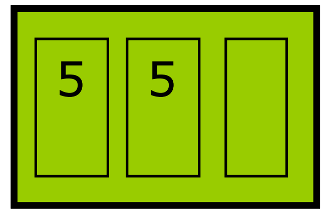


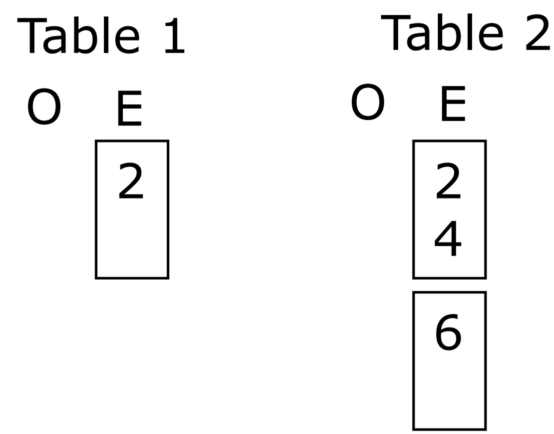
Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R

M=1
$B_1=B_2=2$
p=2



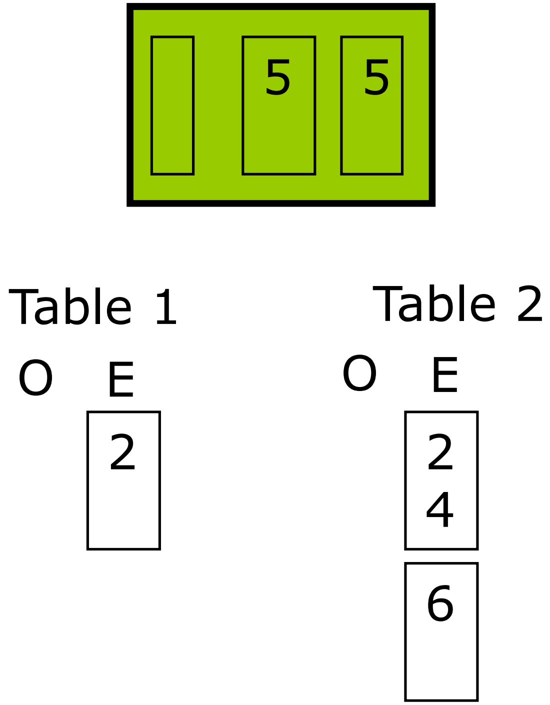


Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R

$M=1$
$B_1=B_2=2$
$p=2$



Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R

M=1
$B_1=B_2=2$
p=2

2	2 4	5

Table 1

O E

Table 2

O E

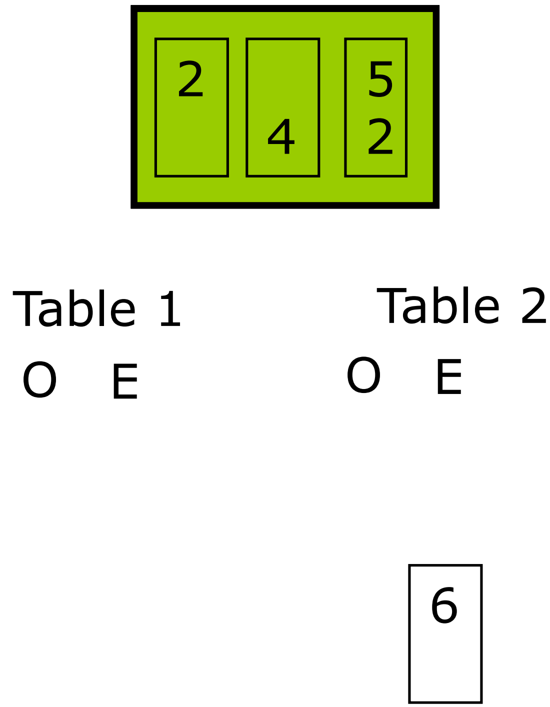
6

Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R

$M=1$
$B_1=B_2=2$
$p=2$



Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R

M=1
$B_1=B_2=2$
p=2

2 4

5
2

Table 1

O E

Table 2

O E

6

Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R

M=1
$B_1=B_2=2$
p=2

2

5
2

Table 1

O E

Table 2

O E

6

Hash Join (II)

Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R
R

M=1
B_1=B_2=2
p=2

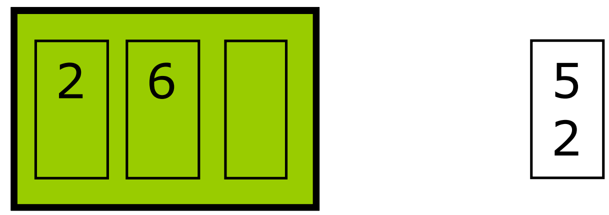


Table 1

O E

Table 2

O E

Hash Join (II)

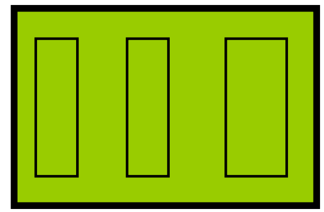
Accesses

R
R
W
W
W
R
W
R
W
W
R
R
R
R
R
R

M=1

$B_1=B_2=2$

p=2



5
2

Table 1

O E

Table 2

O E

Summary table

	No index	B+	Hash	Clustered	Clustered structure
All tuples	Scan				
One tuple		Go through index Go to table	Apply function Go to bucket Go to table	Go through index Go to table	
Several tuples		Go through index Follow leaves Go to table		Go through index Go to table Scan table	
Join	Block Nested Loops Or Hash Join	Row Nested Loops	Row Nested Loops	Row Nested Loops Or Sort-Match	Scan

Summary

- Cost-based optimization
  - Alternatives in the structures
  - Alternatives in the execution
    - Selection algorithms
    - External sort algorithms
    - Projection algorithms
    - Join algorithms
  - Intermediate results estimation

Bibliography

- G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, 1998
- R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill, 3rd Edition, 2003
- S. Lightstone, T. Teorey and T. Nadeau. *Physical Database Design*. Morgan Kaufmann, 2007
- J. Sistac. *Sistemes de Gestió de Bases de Dades*. Editorial UOC, 2002
- Y. Ioannidis. *Query Optimization*. ACM Computing Surveys, vol. 28, num. 1, March 1996.
- J. Lewis. *Cost-Based Oracle Fundamentals*. Apress, 2006