Query Optimization Costs-II

Knowledge objectives

1. Enumerate five join algorithms
2. Explain the prerequisites of each join algorithm (i.e. Row Nested Loops, Block Nested Loops, Hash Join, Sort-Match)
3. Write the pseudo-code of five join algorithms

Understanding objectives

1. Identify when a join algorithm can or cannot be used in a given query or process tree
2. Given the statistics of the tables, estimate the cost of a join using each of the algorithms

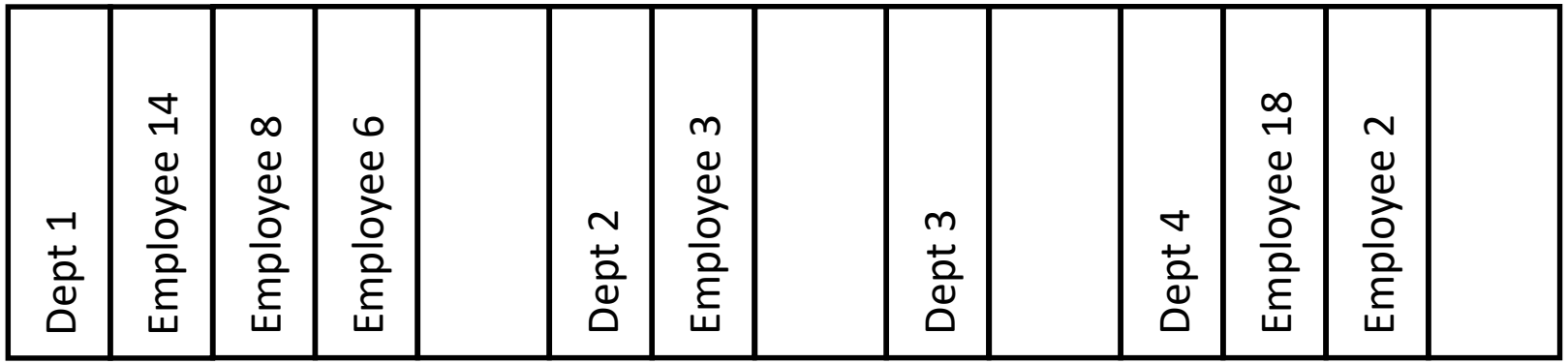
Clustered structure

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



Nested Loops

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
```

- 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: $B_R + |R| \cdot (h_S + 1 + (k-1)/((2/3)R_S))$
  - Hash: $B_R + |R| \cdot (1 + k)$
- Considerations
  - It is only useful if there is an index over the join attribute
    - The table with the index is the internal one
      - If both have an index, we can choose which is the internal one (we must compare both costs)
      - If, according to the process tree, we already performed some operation over the table, its index cannot be used anymore
  - A hash index can only be used for equi-join
  - B_R is the real number of blocks (taking into account a possible cluster)

$u = \%load \cdot 2d = (2/3) \cdot 2d$

$h = \lceil \log_u |T| \rceil - 1$

k = average appearance of each value of R.A in S.A

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 a possible cluster)

Block Nested Loops (II)

Hash-Join

One-pass

Two-pass

One-pass hash Join

- Algorithm
  - Build phase

```
Create an empty hash table
for each block of S
        read block of S into a memory page
        for each tuple t in the read page
            register it in the hash table under key h(t.A)
        endForEach
endForEach
```

  - Probe phase

```
for each block of R
        read block of R into a memory page
        for each tuple t in the read page
            go to the hash table using the key h(t.A)
            if there is any tuple satisfying the join condition
                generate result
            endIf
        endForEach
endForEach
```

- Cost, with M+2 memory pages
  - $B_R + B_S$
- Considerations
  - A hash index is not required to exist a priori
    - It is built in the first phase
  - It can only be used for equi-join
  - It is not symmetric
    - It chooses the smaller table to build the hash table (hence, it is in the internal loop during the probing phase)
  - It requires $B_{Smaller} \leq M$ (so that its hash index fits in memory)

Two-pass hash Join (I)

- Algorithm

  Partition R into *p* parts using a hash function
  Partition S into *p* parts using the same hash function
  Use One-pass hash join *p* times (part by part)

- Cost, with M+2 memory pages
  - $2B_R + 2B_S + B_R + B_S$
    - B_R and B_S are the real number of blocks (taking into account a possible cluster)
      - Beware that after partitioning, the tables have no empty space anymore
- Considerations
  - We must guarantee that each part of S fits into M memory pages later, so we take $p=\lceil B_{Smaller}/M \rceil$
    - Thus, the size of each part $B_{Smaller}/p$ is M pages
      - Assuming the hash function results in a uniform distribution of values
  - Assumes $M<B_{Smaller} \leq M^2+M$
    - In this case $1<p\leq M+1$ (notice that $\lceil M/M \rceil=1$ and $\lceil (M^2+M)/M \rceil=M+1$)
      - If $B_{Smaller} \leq M$ (i.e., $p = 1$), then we use One-pass
      - If $B_{Smaller} > M^2+M$ (i.e., $p>M+1$), then Sort-Match will be preferred

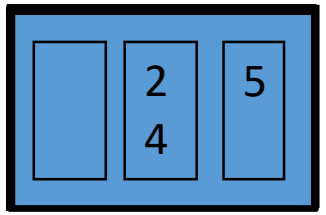
Two-pass hash Join (II)

Pass 1

Pass 2

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

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



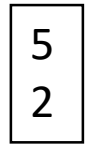


Table 1

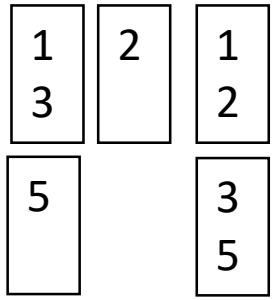
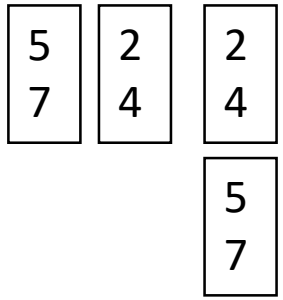


Table 2 (hash)



Sort-Match

Sort-Match (I)

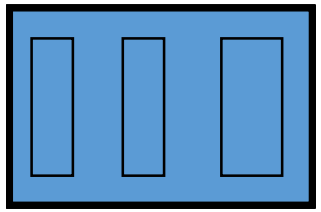
- Algorithm (for equi-join on an attribute A which is unique, at least, in R)

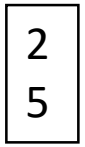
```
Sort R and S (if necessary)
t_R:=first(R); t_S:=first(S);
while not (end(R) or end(S))
    if (t_R[A]<t_S[A]) t_R:=next(R);
    elsIf (t_R[A]>t_S[A]) t_S:=next(S);
    else generate result from t_R and t_S; t_S:=next(S);
    endIf
endWhile
```

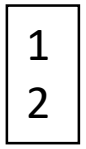
- Cost, with M+1 memory pages
  - Sorting R (same for S):
    - If sorted: 0
    - Elsif $B_R \leq M$: $2B_R$
    - Else: $2B_R \cdot \lceil \log_M B_R \rceil$
  - Merging R and S: $B_R + B_S$
- Considerations
  - It is only useful for equi-join, <-join, >-join and anti-join
    - The given cost corresponds to equi-join and anti-join
  - It is of special interest when at least one table is Clustered (or somehow sorted by the join attribute)
    - If both tables are pre-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 a possible cluster)
    - Beware that after sorting, the auxiliary tables have no empty space anymore

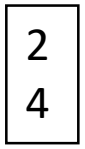
Sort-Match (II)

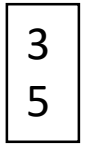
R
R
R
R

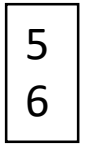












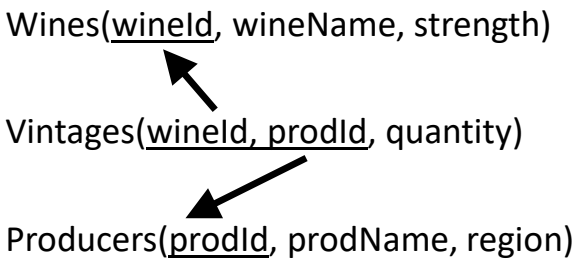
Example of cost-based optimization

Cost-based optimization steps

1. Generate alternatives in the search space
   a. Join order
   b. Potential algorithms
   c. Available structures (access path)
   d. ~~Materialization or not of intermediate results~~
      - We will assume that they are always materialized
2. Evaluate those alternatives
   a. Intermediate results cardinality and size estimation
   b. Cost estimation
3. Choose the best option
4. Generate the corresponding access plan

Example of cost-based optimization (I)

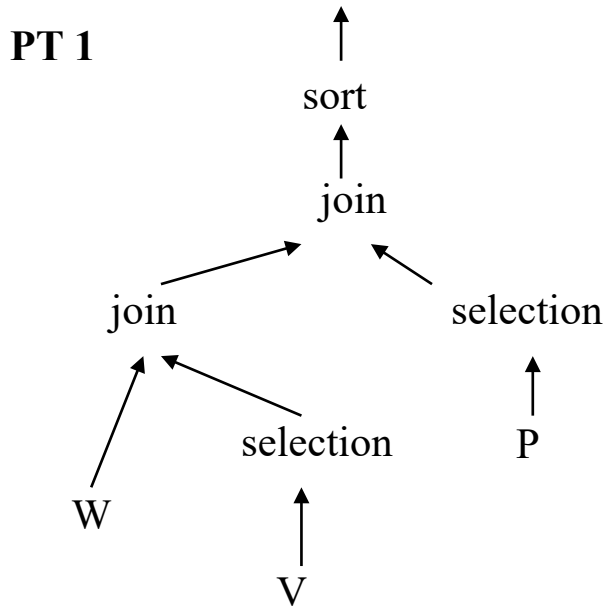
- The tables have the following structures:
  - Producers
    - Clustered by prodId
    - B^+ by region
  - Wines
    - Clustered by wineId
  - Vintages
    - Clustered by wineId and prodId
- We have the following statistics:
  - Tables (extra space due to being clustered needs to be added)
    - $|P|=10000$ $R_P=12$ $B_P=834$
    - $|W|=5000$ $R_W=10$ $B_W=500$
    - $|V|=100000$ $R_V=20$ $B_V=5000$
  - Attributes
    - prodId, wineID and strength: length=5 bytes
    - ndist(region)=30
    - min(quantity)=10 max(quantity)=500
    - ndist(strength)=100
- Moreover, we know that
  - There are 500 useful bytes per intermediate disk block
  - Each table is in a different file (there is no Clustered Structure)
  - Cost of accessing disk blocks is 1 second (D=1)
  - Cost of CPU processing is negligible (C=0)
  - The order of B-trees is 75
  - The DBMS can use:
    - Block Nested Loops (with 6 memory pages, M=4)
    - Row Nested Loops
    - Sort Match (with 3 memory pages for sorting, M=2)
  - We will not change the order of operations coming from syntactic optimization

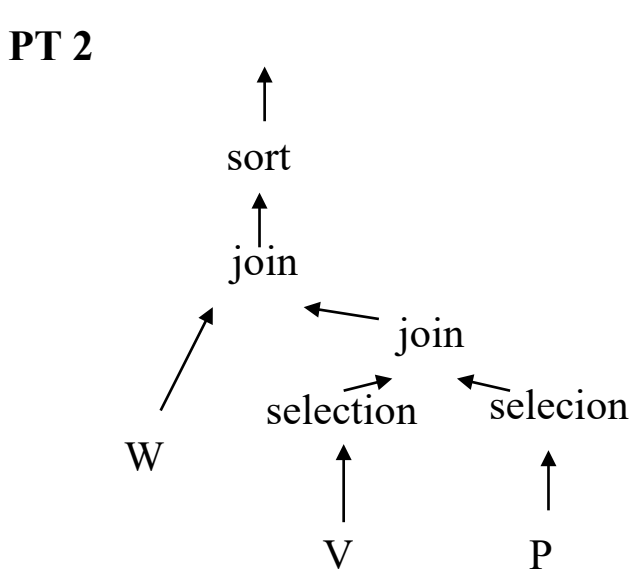


Example of cost-based optimization (II)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```





Example of cost-based optimization (III)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT1/PT2

- **Selection over V: V'**

  Record length V'= 5+5=10 bytes

  SF(quantity>100)=

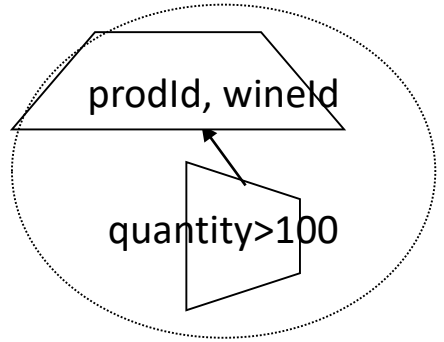
  = (max(quantity)-100)/(max(quantity)-min(quantity)) =

  = 0.81632

  $|V'| = SF^*|V| = 0.81632*100{,}000 = 81{,}632$

  $R_{V'} = \lfloor 500/10 \rfloor = 50$ records/block

  $B_{V'} = \lceil 81{,}632/50 \rceil = 1{,}633$ blocks



- **Selection over P: P'**

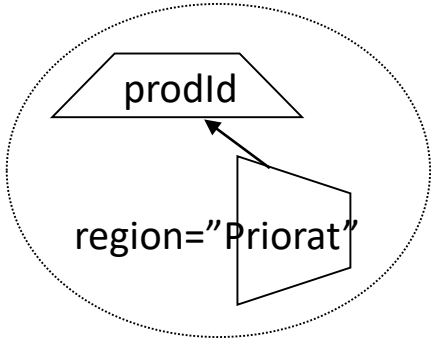
  Record length P'= 5 bytes

  SF(region="Priorat")=1/ndist(region)= 1/30

  $|P'| = SF^*|P| = 10000/30 = 333$

  $R_{P'} = \lfloor 500/5 \rfloor = 100$ records/block

  $B_{P'} = \lceil 333/100 \rceil = 4$ blocks



Example of cost-based optimization (IV)

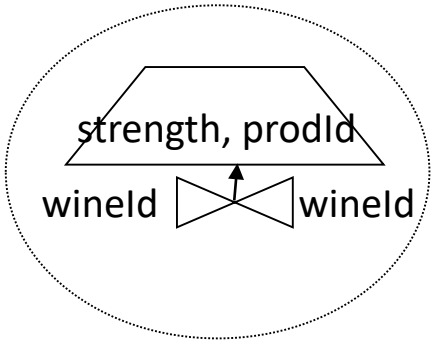
- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT1

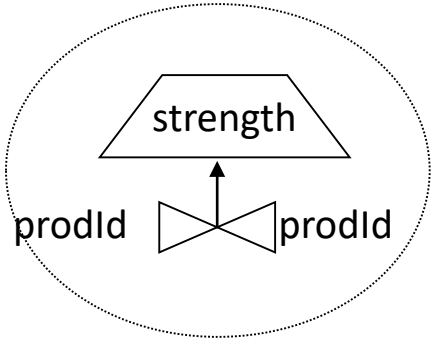
- **Join between W and V': WV'**

  Record length WV' = 5+5 bytes
  SF = 1/|W| = 1/5000
  |WV'| = SF*|W|*|V'|= |V'|= 81,632
  $R_{WV'} = \lfloor 500/10 \rfloor = 50$ records/block
  $B_{WV'} = \lceil 81{,}632/50 \rceil = 1{,}633$ blocks



- **Join between WV' and P': WV'P'** (if quantity and region are independent)

  Record length WV'P' = 5 bytes
  SF(WV'*P') = (1/|P'|)*(1/30)= 10^{-4}
  |WV'P'| = SF*|WV'|*|P'|= 10^{-4}*|WV'|*|P'|= 2,721
  $R_{WV'P'} = \lfloor 500/5 \rfloor = 100$ records/block
  $B_{WV'P'} = \lceil 2721/100 \rceil = 28$ blocks



Example of cost-based optimization (V)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT2

- **Join between V' and P': V'P'** (if quantity and region independent)

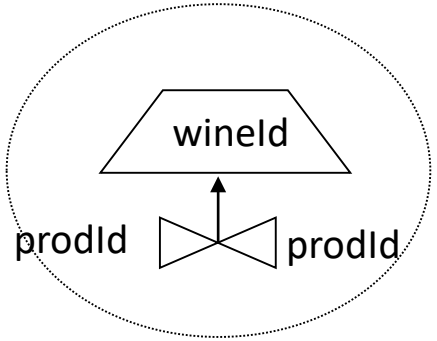
  Record length V'P' = 5 bytes

  $SF(V'*P') = (1/30)*(1/|P'|) = 10^{-4}$

  $|V'P'| = SF*|V'|*|P'| = 10^{-4}*|V'|*|P'| = 2{,}721$

  $R_{V'P'} = \lfloor 500/5 \rfloor = 100$ records/block

  $B_{V'P'} = \lceil 2721/100 \rceil = 28$ blocks



- **Join between W and V'P': WV'P'**

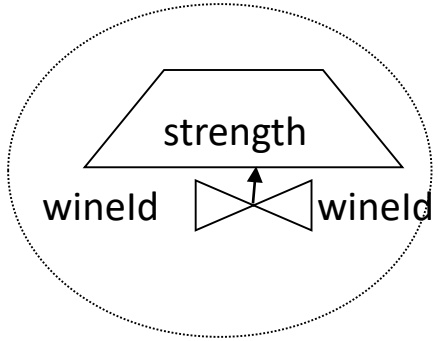
  Record length WV'P = 5 bytes

  SF = 1/|W|

  $|WV'P'| = FS*|W|*|V'P'| = |V'P'| = 2{,}721$

  $R_{WV'P'} = \lfloor 500/5 \rfloor = 100$ records/block

  $B_{WV'P'} = \lceil 2721/100 \rceil = 28$ blocks



Example of cost-based optimization (VI)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

PT1/PT2

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

- **Final result: O**

  Record length O = 5 bytes

  $|O|$ = ndist(strength) = 100

  $R_o = \lfloor 500/5 \rfloor = 100$ records/block

  $B_o = \lceil 100/100 \rceil = 1$ blocks

Example of cost-based optimization (VII)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- **Step 2b: Cost estimation for each algorithm**
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT1/PT2

- **Selection over V: V'**

  Available access paths: No index

  $cost_{Scan}(V') = \lceil 1.5B_V \rceil = \lceil 1.5*5{,}000 \rceil = 7{,}500$

  Choose Scan

- **Selection over P: P'**

  Available access paths: B+ and No index

  $cost_{Scan}(P') = \lceil 1.5*B_P \rceil = \lceil 1.5*834 \rceil = 1{,}251$

  $cost_{B+}(P') = \lceil \log_{100}|P| \rceil - 1 + SF(region="Priorat")*|P| + ((SF(region="Priorat")*|P|-1)/100) = 1+333+332/100 = 337$

  Choose B+

- **Sort of WV'P': O**

  $cost_{MergeSort}(O) = 2B_{WV'P'} \cdot \lceil \log_M(B_{WV'P'}) \rceil - B_{WV'P'} = 2 \cdot 28 \cdot \lceil \log_2(28) \rceil - 28 = 252$

Example of cost-based optimization (VIII)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- **Step 2b: Cost estimation for each algorithm**
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT1

- **Join between W and V': WV'**

  Available algorithms:

  Block Nested Loops

  $\lceil 1.5 \cdot B_W \rceil < B_{V'}$ (use commutative property of joins)

  $cost_{NestedLoop}(WV') = \lceil 1.5B_W \rceil + \lceil 1.5B_W / M \rceil * B_{V'} = \lceil 1.5*500 \rceil + \lceil 1.5*500/4 \rceil * 1633 = 307{,}754$

  Row Nested Loops

  Yes, we do look for attributes of W

  V' does not use extra space any more for being ordered

  $cost_{RowNestedLoops}(WV') = B_{V'} + |V'| * (\lceil \log_{100} |W| \rceil - 1 + 1 + (1.5(k-1)/10)) = 1{,}633 + 81{,}632 * (\lceil \log_{100} 5{,}000 \rceil - 1 + 1) = 164{,}897$

  Sort-Match

  W is ordered by wineID, V' is still ordered by wineId and prodId

  $cost_{SortMatch}(WV') = \lceil 1.5B_W \rceil + B_{V'} = \lceil 1.5*500 \rceil + 1{,}633 = 2{,}383$

  Choose Sort-Match

Example of cost-based optimization (IX)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- **Step 2b: Cost estimation for each algorithm**
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT1

- **Join between WV' and P': WV'P'**

  Available algorithms:

  Block Nested Loops

  $B_{P'} < B_{WV'}$ (use commutative property of joins)

  $cost_{NestedLoop}(WV'P') = B_{P'} + \lceil B_{P'} / M \rceil * B_{WV'} = 4 + \lceil 4/4 \rceil * 1{,}633 = 1{,}637$

  Sort Match

  Neither WV' nor P' are ordered by prodId

  $cost_{SortMatch}(WV'P') = 2*B_{WV'} * \lceil \log_2 B_{WV'} \rceil + 2*B_{P'} * \lceil \log_2 B_{P'} \rceil + B_{WV'} + B_{P'} = 2*1{,}633*11 + 2*4*2 + 1633 + 4 = 37{,}579$

  Choose Nested Loops

Example of cost-based optimization (X)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- **Step 2b: Cost estimation for each algorithm**
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT2

- **Join between V' y P': V'P'**

  Available algorithms:

  Block Nested Loops

  $B_{P'} < B_{V'}$ (use commutative property of joins)

  $cost_{NestedLoop}(V'P') = B_{P'} + \lceil B_{P'}/M \rceil * B_{V'} = 4 + \lceil 4/4 \rceil * 1{,}633 = 1{,}637$

  Sort Match

  Neither V' nor P' are ordered by prodId

  $cost_{SortMatch}(V'P') = 2*B_{V'}*\lceil \log_2 B_{V'} \rceil + 2*B_{P'}*\lceil \log_2 B_{P'} \rceil + B_{V'} + B_{P'} = 2*1{,}633*11 + 2*4*2 + 1{,}633 + 4 = 37{,}579$

  Choose Nested Loops

Example of cost-based optimization (XI)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- **Step 2b: Cost estimation for each algorithm**
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```

PT2

- **Join between W y V'P': WV'P'**

  Available algorithms:

  Block Nested Loops

  $B_{V'P'} < \lceil 1.5B_W \rceil$ (use commutative property of joins)

  cost $_{NestedLoop}(WV'P') = B_{V'P'} + \lceil B_{V'P'} / M \rceil * \lceil 1.5B_W \rceil = 28 + \lceil 28/4 \rceil * \lceil 1.5*500 \rceil = 5278$

  Row Nested Loops

  Yes, we look for attributes of W

  cost $_{RowNestedLoops}(WV'P') = B_{V'P'} + |V'P'| * (\lceil \log_{100} |W| \rceil - 1 + 1 + (1.5(k-1)/10)) = 28 + 2{,}721 * (\lceil \log_{100} 5{,}000 \rceil - 1 + 1) = 5{,}470$

  Sort-Match

  W is sorted by wineId, V'P' is not sorted by wineId

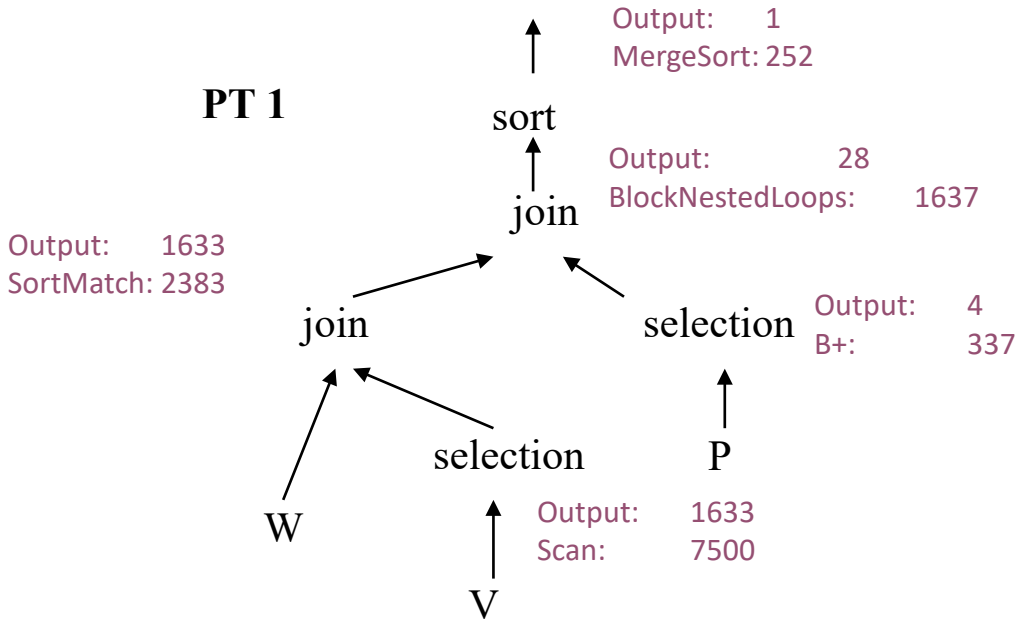
  cost $_{SortMatch}(WV'P') = 2B_{V'P'} \lceil \log_2 B_{V'P'} \rceil + \lceil 1.5B_W \rceil + B_{V'P'} = 2*28* \lceil \log_2 28 \rceil + \lceil 1.5*500 \rceil + 28 = 1{,}058$

  Choose Sort-Match

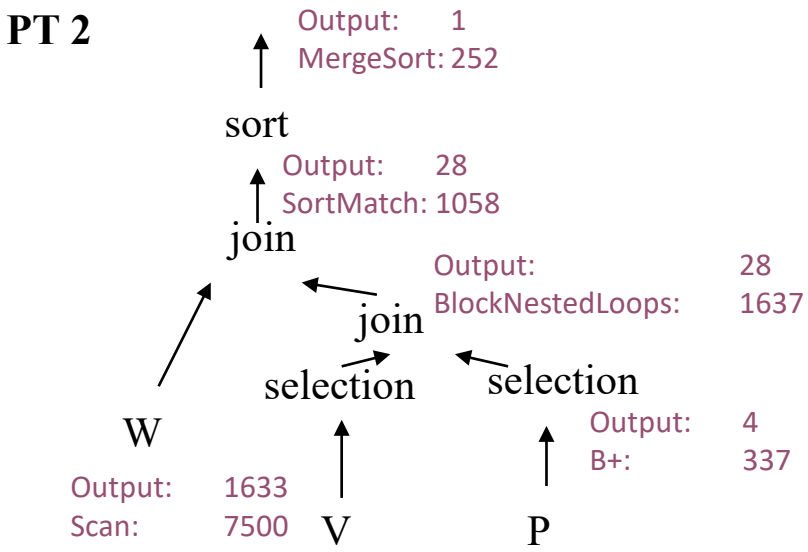
Example of cost-based optimization (XII)

- Step 1: Generate alternatives
- Step 2a: Intermediate results estimation
- Step 2b: Cost estimation for each algorithm
- Step 3: Choose the best option

```
SELECT DISTINCT w.strength
FROM wines w, producers p, vintages v
WHERE v.wineId=w.wineId
        AND p.prodId=v.prodId
        AND p.region="Priorat"
        AND v.quantity>100;
```



$Cost_{PT1}$: 15,408



$Cost_{PT2}$: 12,478

Closing

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 estimation
  - Join algorithms
    - Clustered structure
    - Nested loops
      - Row
      - Block
    - Hash
      - One-pass
      - Two-pass
    - Sort-Match

Bibliography

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