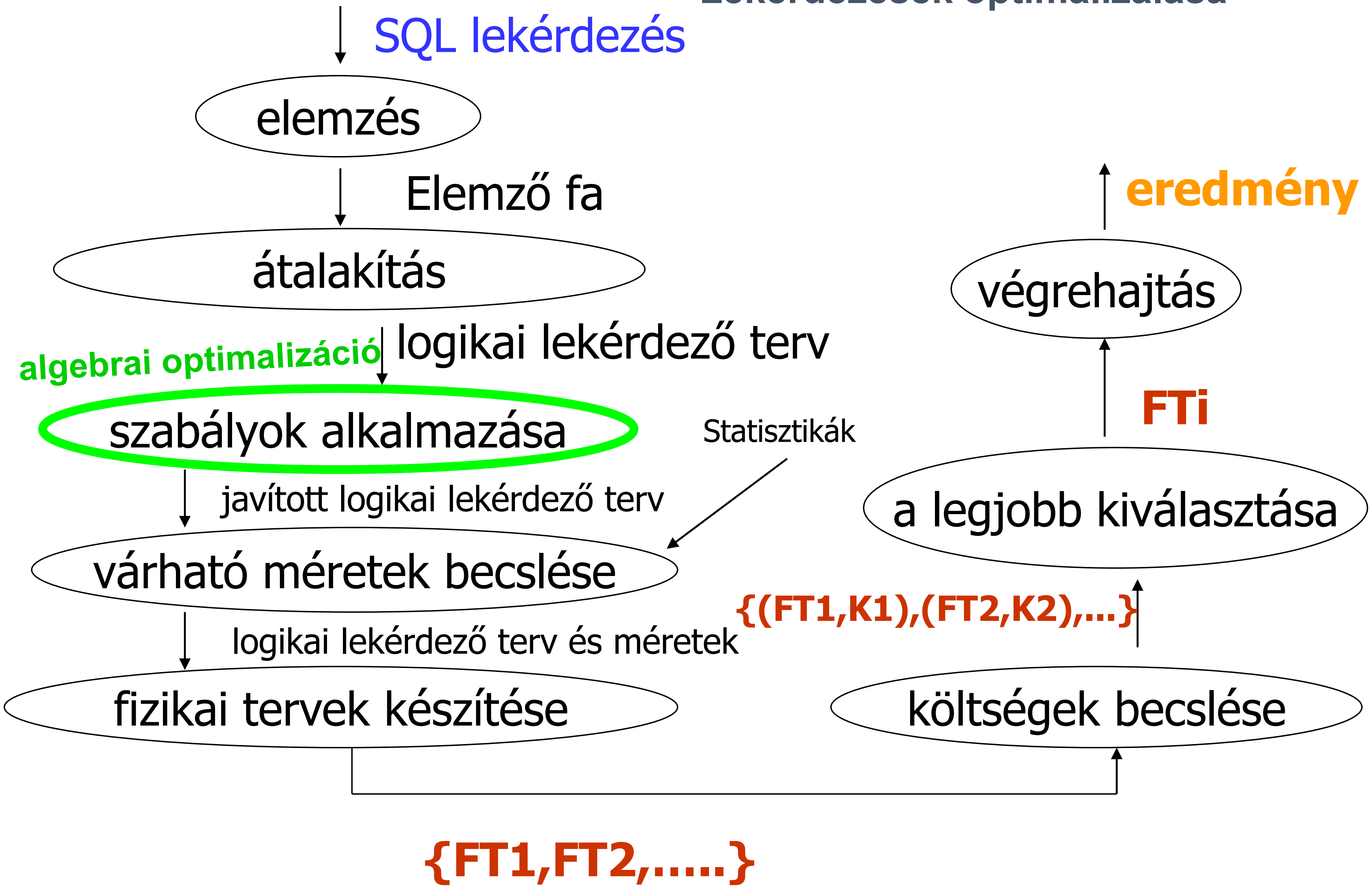
Több tábla összekapcsolása


SQL lekérdezés
elemzés
Elemző fa
átalakítás
algebrai optimalizáció
logikai lekérdező terv
szabályok alkalmazása
Statisztikák
javított logikai lekérdező terv
várható méretek becslése
logikai lekérdező terv és méretek
fizikai tervek készítése
{FT1,FT2,.....}
költségek becslése
{(FT1,K1),(FT2,K2),...}
a legjobb kiválasztása
FTi
végrehajtás
eredmény

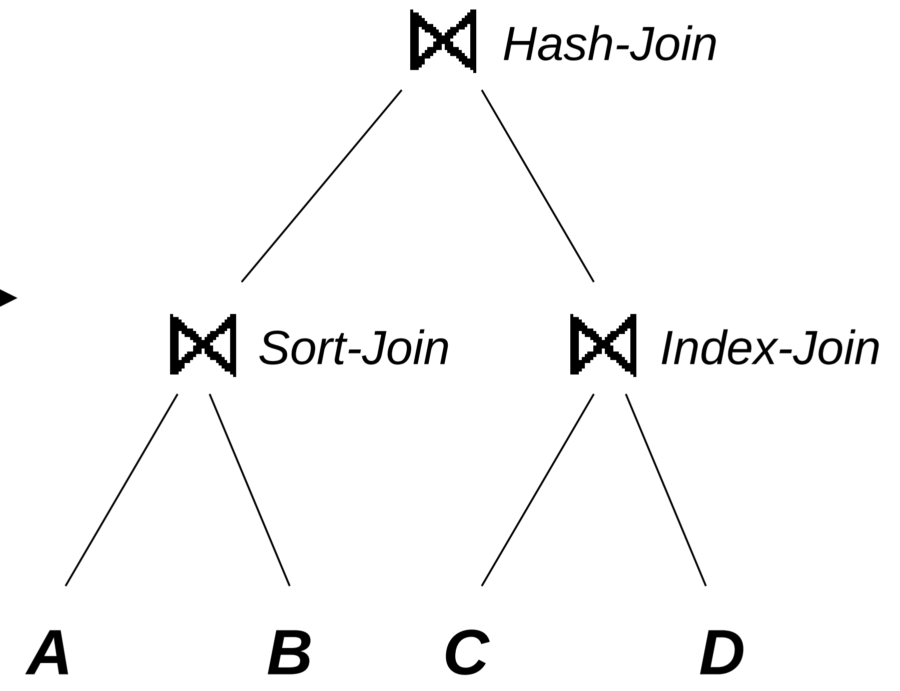
Algebrai optimalizáció

- könyv(sorszám,író,könyvcím)
  - **kv(s,i,kc)**
- kölcsönző(azonosító,név,lakcím)
  - **kő(a,n,lc)**
- kölcsönzés(sorszám,azonosító,dátum)
  - **ks(s,a,d)**
- Milyen című könyveket kölcsönöztek ki 2007-től kezdve?
- $\Pi_{kc}(\sigma_{d>='2007.01.01'}(kv|\times|kő|\times|ks))$
- Az összekapcsolásokat valamilyen sorrendben kifejezzük az alapműveletekkel:

$$\Pi_{kc}(\sigma_{d>='2007.01.01'}(\Pi_{kv.s,i,kc,kő.a,n,lc,d}(\sigma_{kv.s=ks.s \wedge kő.a=ks.a}(kv\times(kő\times ks)))))$$

What is the best way to join *n* relations?

SELECT …
FROM A, B, C, D
WHERE A.x = B.y
AND C.z = D.z



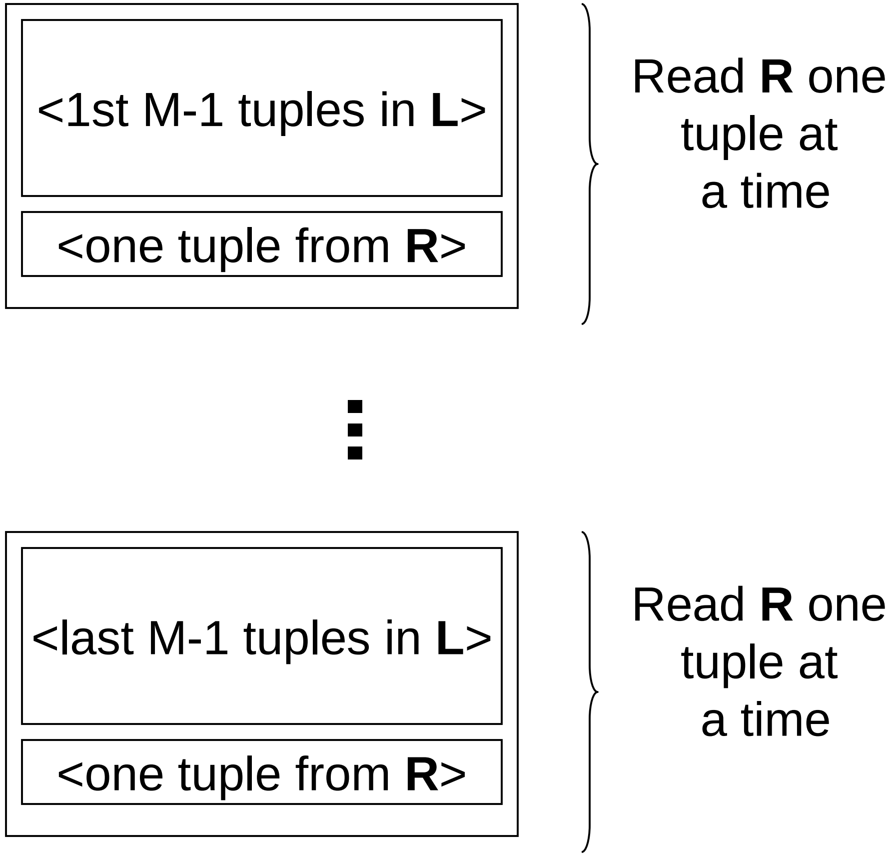
Issues to consider for 2-way Joins

- Join attributes sorted or indexed?
- Can either relation fit into memory?
  - Yes, evaluate in a single pass
  - No, require $Log_M B$ passes. M is #of buffer pages and B is # of pages occupied by smaller of the two relations
- Algorithms (nested loop, sort, hash, index)

$L \bowtie R$

Nested Loop Join

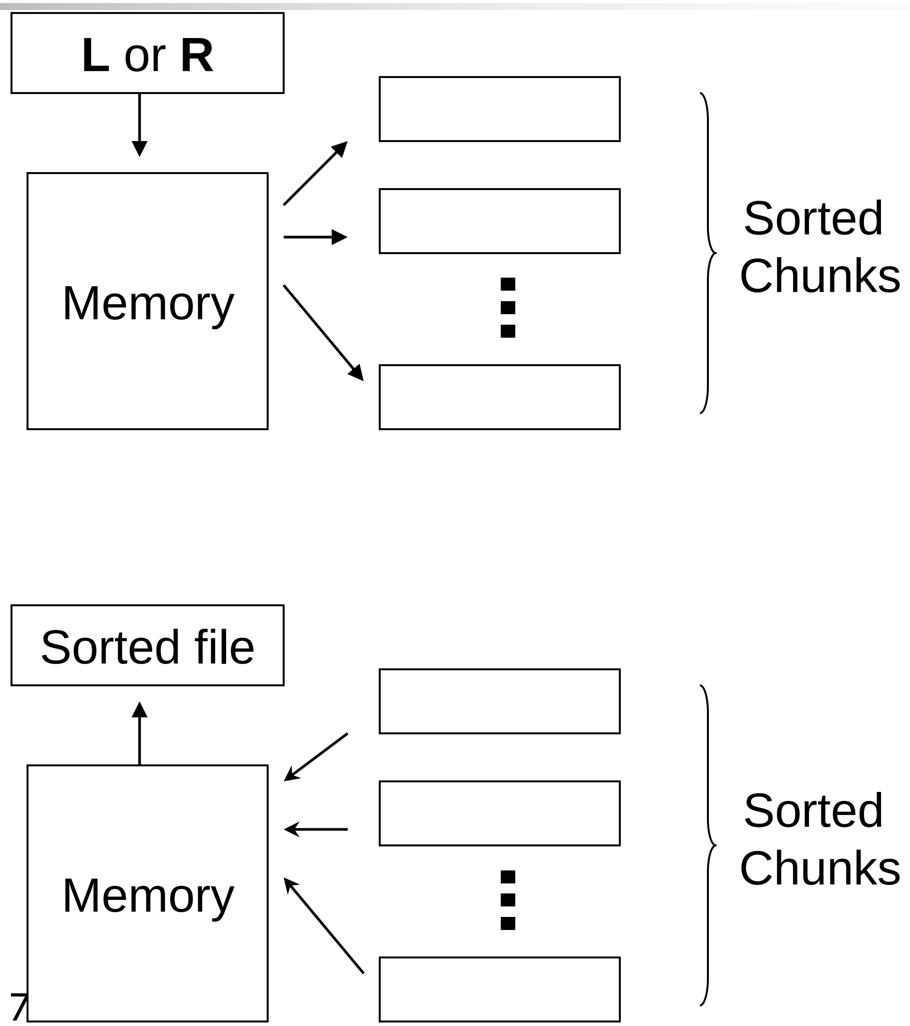
- L is left/outer relation
- Useful if no index, and not sorted on the join attribute
- Read as many pages of L as possible into memory
- Single pass over L, B(L)/(M-1) passes over R



Sort Merge Join

$L \bowtie R$

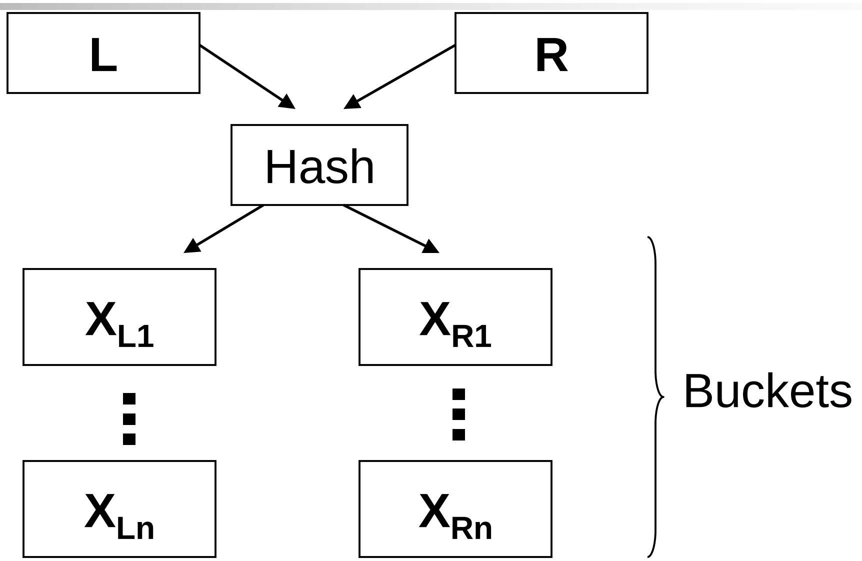
- Useful if either relation is sorted. Result sorted on join attribute.
- Divide relation into M sized chunks
- Sort the each chunk in memory and write to disk
- Merge sorted chunks

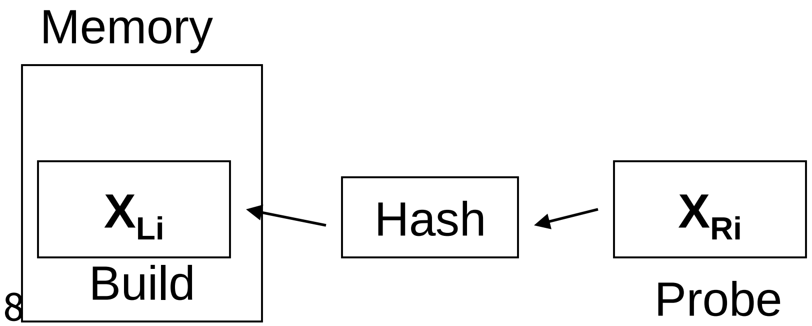


$L \bowtie R$

Hash Join

- Hash (using join attribute as key) tuples in **L** and **R** into respective buckets
- Join by matching tuples in the corresponding buckets of **L** and **R**
- Use **L** as build relation and **R** as probe relation



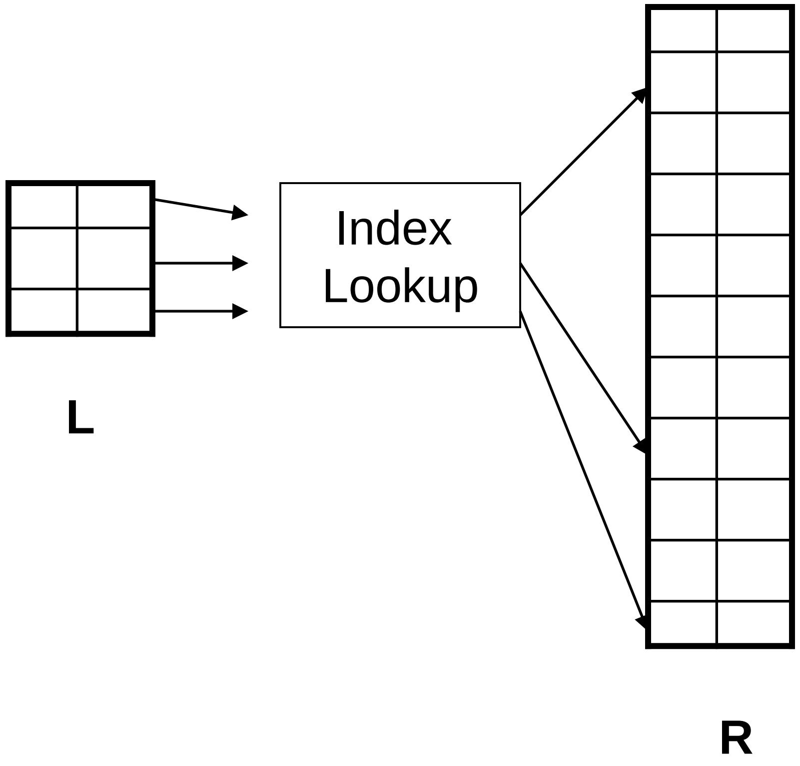


$L \bowtie R$

Index Join

- Join attribute is indexed in **R**
- Match tuples in **R** by performing an index lookup for each tuple in **L**
- If clustered b-tree index, can perform sort-join using only the index

- Costs:
  - $B_L + T_L * C$
    - C the cost of index-based selection of inner relation R
    - $C = B_R/I$ If R is clustered
    - $C = T_R/I$ If R is not clustered



Ordering N-way Joins

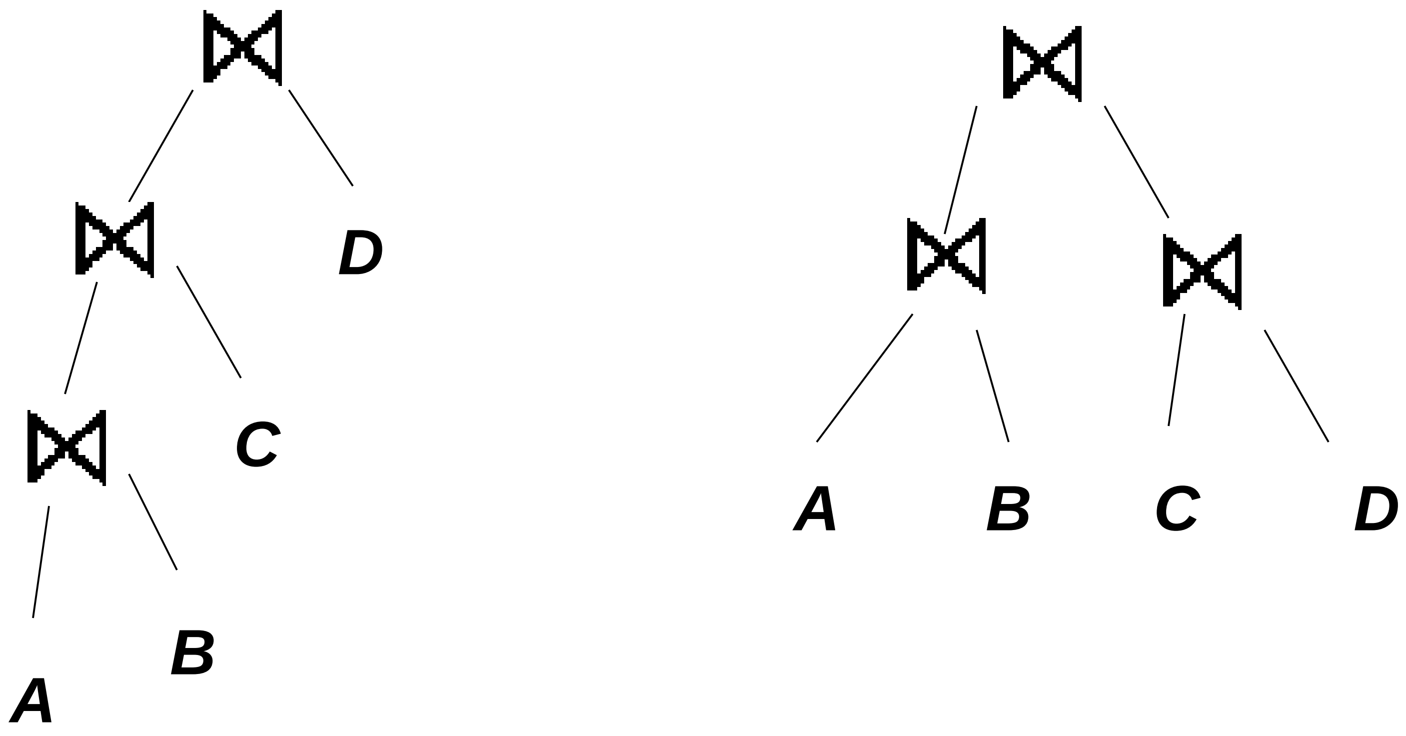
- Joins are commutative and associative
  - $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$
- Choose order which minimizes the sum of the sizes of intermediate results
  - Likely I/O and computationally efficient
  - If the result of $A \bowtie B$ is smaller than $B \bowtie C$, then choose $(A \bowtie B) \bowtie C$
- Alternative criteria: disk accesses, CPU, response time, network

Ordering N-way Joins

- Choose the shape of the join tree
  - Equivalent to number of ways to parenthesize n-way joins
  - Recurrence: $T(1) = 1$

    $T(n) = \Sigma\ T(i)T(n-i)$, $T(6) = 42$
- Permutation of the leaves
  - $n!$
- For $n = 6$, the number of join trees is $42 \cdot 6!$ Or 30,240

Shape of Join Tree



Left-deep Tree ***Bushy Tree***

Shape of Join Tree

- A left-deep (right-deep) tree is a join tree in which the right-hand-side (left-hand-side) is a relation, not an intermediate join result
- Bushy tree is neither left nor right-deep
- Considering only left-deep trees is usually good enough
  - Smaller search space
  - Tend to be efficient for certain join algorithms (order relations from smallest to largest)
  - Allows for pipelining
  - Avoid materializing intermediate results on disk

Searching for the best join plan

- Exhaustively enumerating all possible join order is not feasible (n!)
- Dynamic programming
  - use the best plan for (k-1)-way join to compute the best k-way join
- Greedy heuristic algorithm
  - Iterative dynamic programming

Dynamic Programming

- The best way to join k relations is drawn from k plans in which the left argument is the least cost plan for joining k-1 relations
- BestPlan(A,B,C,D,E) = min of (

  BestPlan(A,B,C,D) $\bowtie$ E,

  BestPlan(A,B,C,E) $\bowtie$ D,

  BestPlan(A,B,D,E) $\bowtie$ C,

  BestPlan(A,C,D,E) $\bowtie$ B,

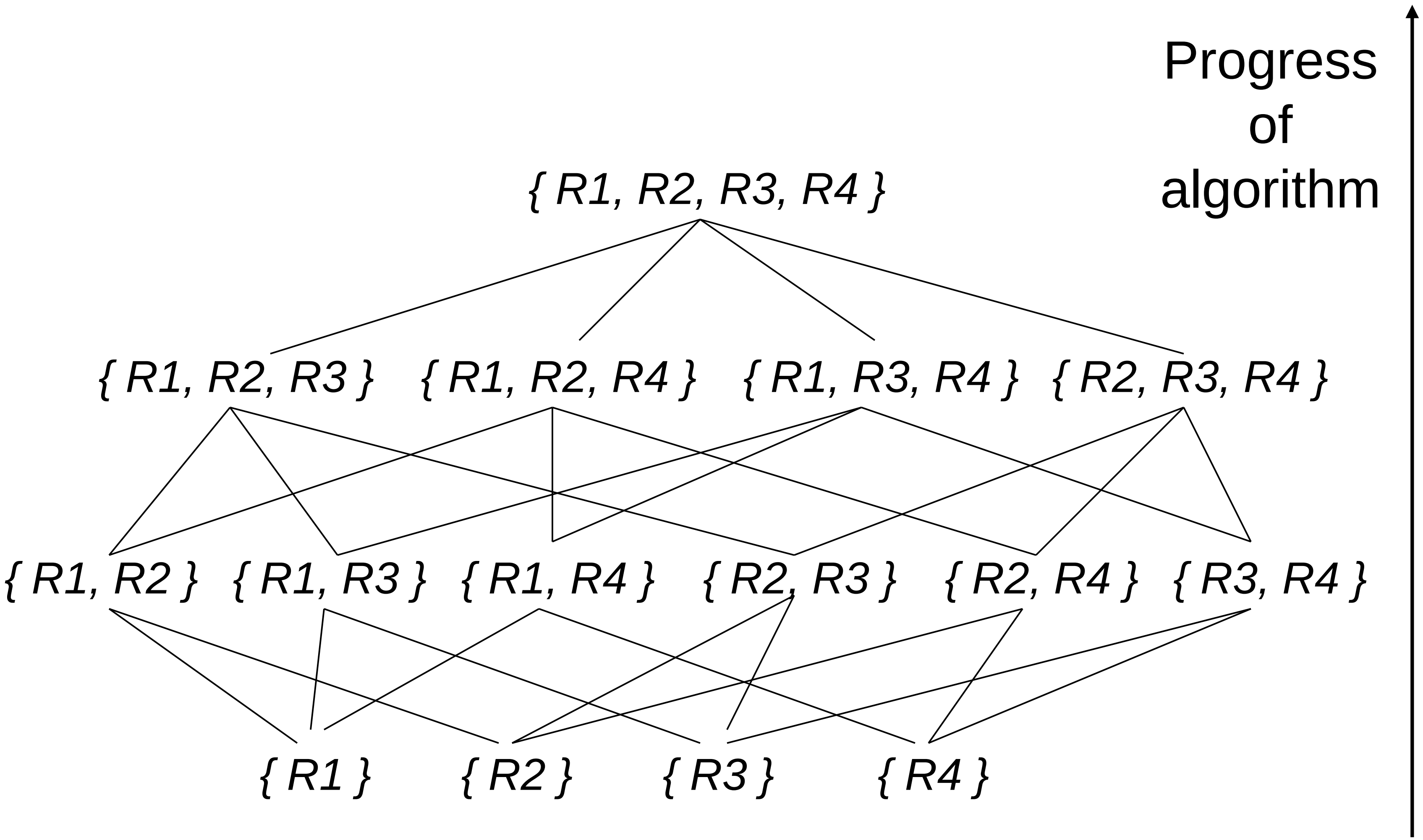
  BestPlan(B,C,D,E) $\bowtie$ A)

Complexity

- Finds optimal join order but must evaluate all 2-way, 3-way, …, n-way joins (n choose k)
- Time $O(n*2^n)$, Space $O(2^n)$
- Exponential complexity, but joins on > 10 relations rare

Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



Notation

OPT ({ $R1, R2, R3$ }):

Cost of optimal plan to join $R1,R2,R3$

T ({ $R1, R2, R3$ }):

Number of tuples in $R1 \bowtie R2 \bowtie R3$

Selinger Algorithm:

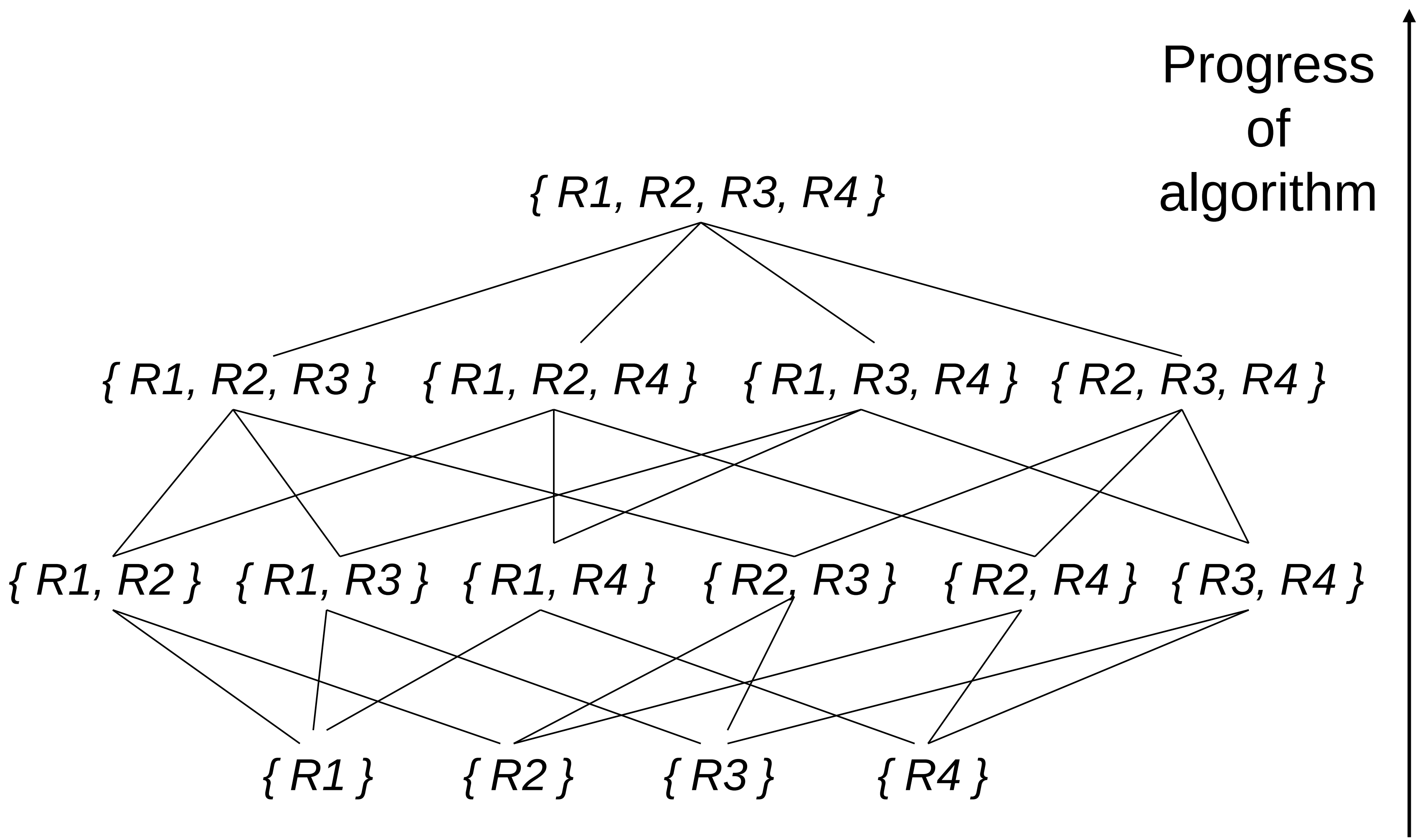
OPT ({ R1, R2, R3 }):

$$
\text{Min} \begin{cases} \text{OPT}(\{R1, R2\}) + T(\{R1, R2\}) + T(R3) \\[2em] \text{OPT}(\{R2, R3\}) + T(\{R2, R3\}) + T(R1) \\[2em] \text{OPT}(\{R1, R3\}) + T(\{R1, R3\}) + T(R2) \end{cases}
$$

Note: Valid only for the simple cost model

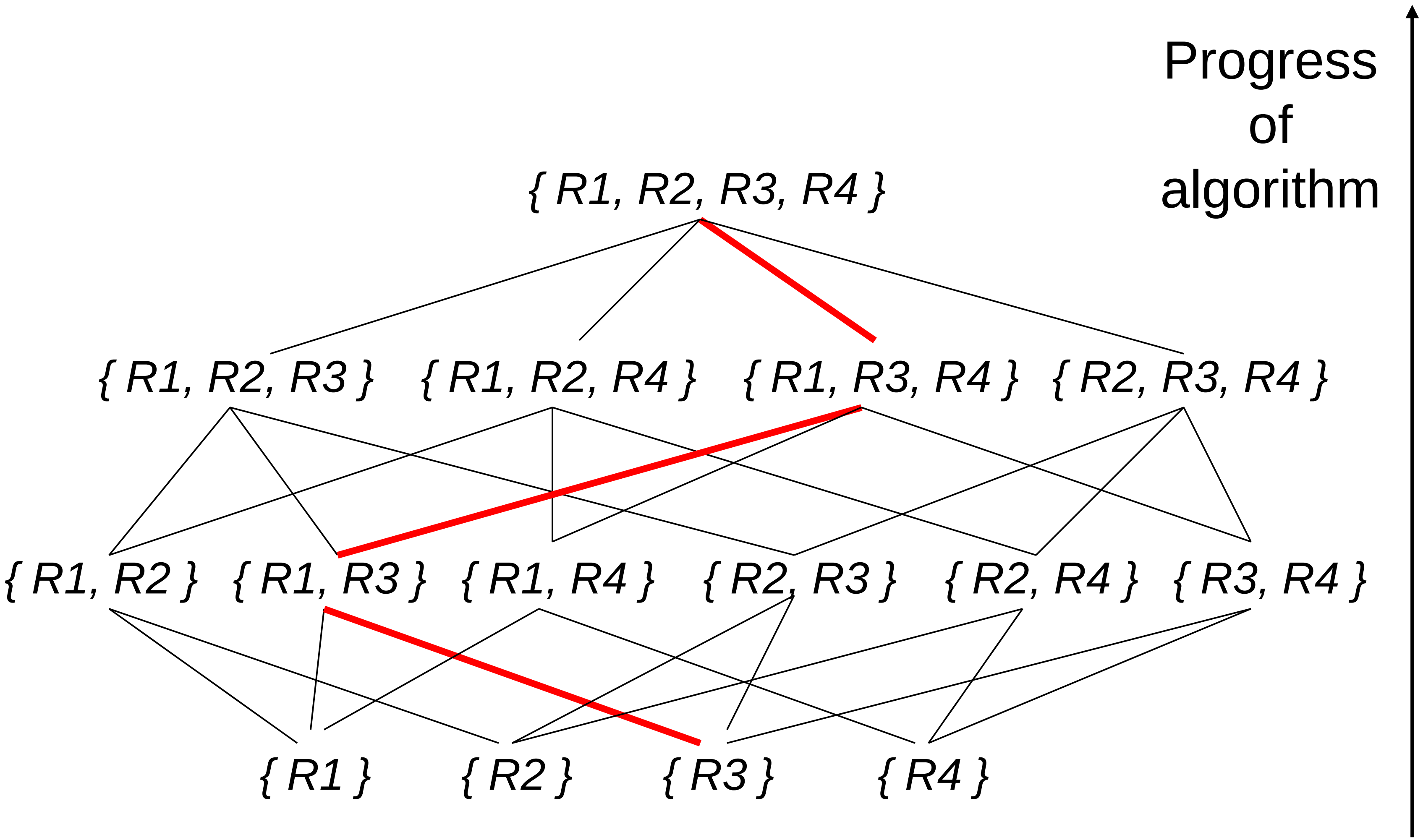
Selinger Algorithm:

Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



Selinger Algorithm:

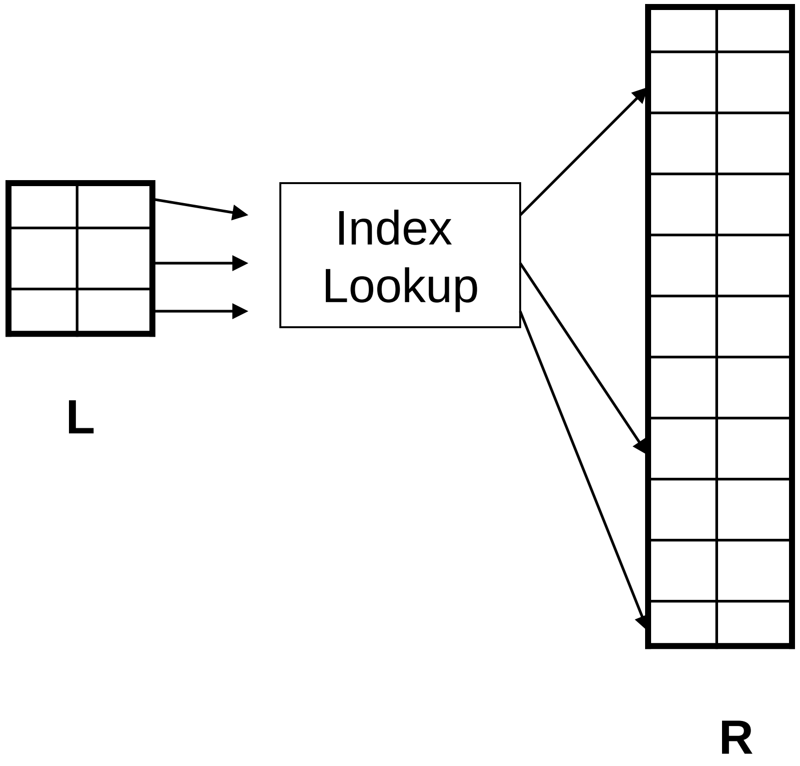
Query: $R1 \bowtie R2 \bowtie R3 \bowtie R4$



$L \bowtie R$

Index Join

- Join attribute is indexed in **R**
- Match tuples in **R** by performing an index lookup for each tuple in **L**
- If clustered b-tree index, can perform sort-join using only the index



- Costs:
  - $B_L + T_l * C$
    - C the cost of index-based selection of inner relation R
    - $C = B_R/I$ If R is clustered
    - $C = T_R/I$ If R is not clustered

9. tétel

A *Q(A,B) JOIN R(B,C) JOIN S(C,D)* háromféle kiszámítási módja és költsége, (feltéve, hogy Q,R,S paraméterei megegyeznek, Q.B-re és S.C-re klaszterindexünk van).

a) balról jobbra,

b) balról jobbra és a memóriában összekapcsolva a harmadik táblával,

c) a középső ténytábla soraihoz kapcsolva a szélső dimenziótáblákat.

Feltevések:

$T_Q = T_R = T_S = T$ (ugyananannyi soruk van)

$B_Q = B_R = B_S = B$ (ugyananannyi helyet foglalnak)

$I_{Q.B} = I_{R.B} = I_{R.C} = I_{S.C} = I$ (a képméretek, vagyis az előforduló értékek száma azonos)

Előzetes számítások

Az alábbiakban kiszámolt értékeket fel fogjuk használni a későbbiekben.
Először nézzük meg, hogyan lehetne előállítani két tábla összekapcsolását *R(A,B) JOIN S(B,C)*-t, ha mindkét táblán van index a közös oszlopra. Az azonos értékekhez tartozó sorokat az indexek alapján olvassuk be a táblákból, majd a memóriában összekapcsoljuk őket. Feltesszük, hogy az összekapcsolandó sorok beférnek a memóriába, vagyis $B_R/I + B_S/I <= M$, valamint, hogy R.B részhalmaza S.B-nek. Egy index segítségével történő beolvasás költsége ≈ a beolvasott blokkok száma, vagyis $B_R/I_{R.B}$ illetve $B_S/I_{R.S}$

A teljes **JOIN művelet I/O költsége** (beolvassuk R-et, majd minden sorához index segítségével S-et. Az alábbi képlet az output kiírásának költségét nem tartalmazza.)
$B_R + T_R * B_S/I_{S.B}$ $I_{R.B}=I_{S.B} = I$ esetén:

(1) $\mathbf{B_R + T_R * B_S/I}$

Hány sora lesz a JOIN-nak?
$T_{R|><|S} = I_{R.B} * (T_R/I_{R.B} * T_S/I_{S.B})$ (az egyes értékekhez tartozó részek direkt szorzata)

Ha feltesszük, hogy $I_{R.B}=I_{S.B} = I$, akkor **a JOIN sorainak száma:**

(2) $\mathbf{T_{R|><|S} = T_R * T_S/I}$

Mekkora méretű lesz az output? (RxS esetén $T_R * B_S + T_S * B_R$ lenne)
Az output mérete:

(3) $\mathbf{(T_R * B_S + T_S * B_R)/I}$

A fenti 3 képletet fogjuk felhasználni a *Q(A,B) JOIN R(B,C) JOIN S(C,D)* kiszámításához.

a) balról jobbra történő kiszámítás

Q(A,B) JOIN R(B,C)-re

Output mérete:	**2*T*B/I**	lásd (3)
Sorok száma:	**T^2/I**	lásd (2)
I/O költség:	**B + T*B/I**	lásd (1)

Használjuk fel a fentieket *Q(A,B) JOIN R(B,C) JOIN S(C,D)* esetén az output és az I/O költség kiszámításához.

Output mérete (3)-ba helyettesítve: $[(T^2/I)*B + (2*T*B/I)*T]/I = 3*T^2*B/I^2$

A teljes JOIN I/O költsége:

Az 1. join költsége **$B + T*B/I$** plusz
Az 1. join kiírása (output mérete): **$2*T*B/I$** plusz
A 2. join költsége **$2*T*B/I + [(T^2/I)*B]/I$** plusz
A teljes output kiírása: **$3*T^2*B/I^2$**
összesen:

a) végeredménye: $B + 5*T*B/I + 4*T^2*B/I^2$

b) balról jobbra és a memóriában összekapcsolva a harmadik táblával,
Megspórolhatjuk az 1. join eredményének kiírását majd újbóli beolvasását, vagyis $2*(2*T*B/I)$-t. Az eredmény ekkor:

b) végeredménye: $B + T*B/I + 4*T^2*B/I^2$

c) végeredménye: $\mathbf{B + 2*T*B/I + 3*T^2*B/I^2}$

Nézzük meg, hogy a b) és c) esetek közül melyik a kisebb költségű. A két költség közötti különbség (b-c): $\mathbf{T^2*B/I^2 - T*B/I}$

Nagyméretű táblák esetén a T/I hányados nagy szám lesz, ezért a négyzetes tag jóval nagyobb lesz, mint a lineáris tag, vagyis a c) módszer a leghatékonyabb.

Ha a c/b arányt tekintjük, akkor azt mondhatjuk, hogy ez az arány ¾-hez tart, ha T/I tart a végtelenbe. Vagyis ha T/I elég nagy, akkor a c költsége nagyjából ¾-e a b-nek.