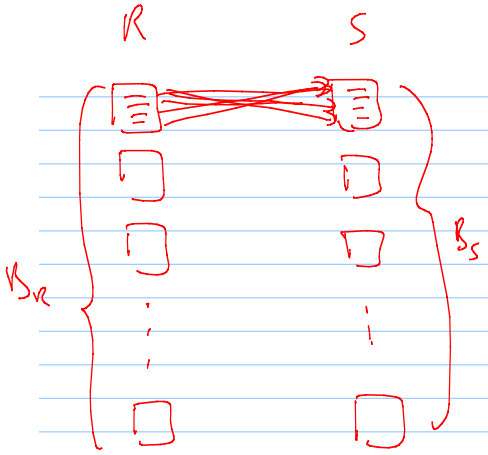
An Improvement: Block Nested-Loop Join

- Evaluate the condition join R $\bowtie_C$ S
- **for each** block B_R **of** R **do begin**
  for each block B_S **of** S **do begin**
  for each tuple t_R **in** B_R **do**
  for each tuple t_S **in** B_S **do**
  check whether pair (t_R, t_S)
  satisfies join condition
  if they do, add $t_R \circ t_S$ to the result
  end end end end
- Also requires no indexes and can be used with any kind of join condition.
- Worst case: db buffer can only hold one block of each relation
  $\Longrightarrow$ $B_R + B_R * B_S$ disk accesses.
- Best case: both relations fit into db buffer
  $\Longrightarrow$ $B_R + B_S$ disk accesses.
- If smaller relation completely fits into db buffer, use that as



$B_R + B_R * B_S$

read R (outer loop)

read S, B_R - many times

(vs. $B_R + N_R * B_S$ before)

CUSTOMERS ⋈ ORDERS

$B_R = 250$

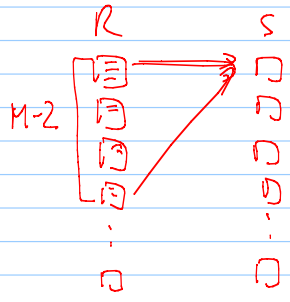
$B_S = 400$

$B_R + B_R * B_S =$

$250 + 250 \cdot 400 = 100,250$ Block I/Os

if you use naive nested loop ⇝ $\gg 2,000,000$

"M-2 trick"



$B_R + \frac{B_R}{M-2} * B_S$

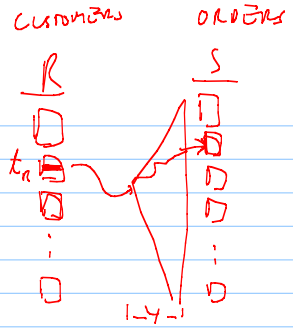
$B_R = 250$

$M-2 = 50$ ⇝ $250 + \frac{250}{50} \cdot 400$

$= 2,250$

Example:

- Compute CUSTOMERS ⋈ ORDERS, with CUSTOMERS as the outer relation.
- Let ORDERS have a primary B^+-tree index on the join-attribute CName, which contains 20 entries per index node
- Since ORDERS has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data records (based on tuple identifier).
- Since $N_{CUSTOMERS}$ is 5,000, the total cost is 250 + 5000 ∗ 5 = 25,250 disk accesses.
- This cost is lower than the 100,250 accesses needed for a block nested-loop join.



$\log_{20} 10{,}000 \simeq 3.1$ $\quad 20^3 = 8000$

$R \bowtie S$

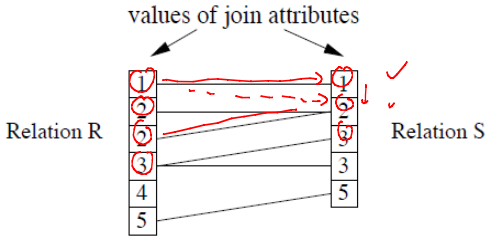
equi-join $R.A = S.B$ ✓

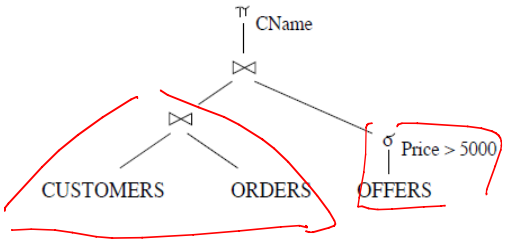
theta-join $R.A < S.B$ maybe

$R.A \neq S.B$ probably not

Sort-Merge Join

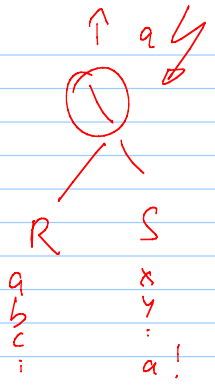
- Basic idea: first sort both relations on join attribute (if not already sorted this way)
- Join steps are similar to the merge stage in the external sort-merge algorithm (discussed later)
- Every pair with same value on join attribute must be matched.



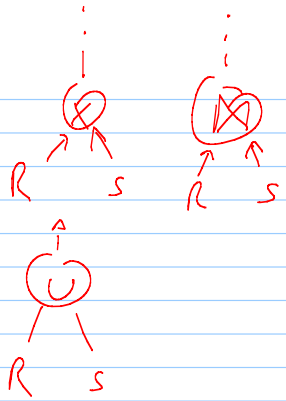


tmp1 := CUSTOMERS ⋈ ORDERS
tmp2 := σ_{Price} (OFFERS)
tmp3 := tmp1 ⋈ tmp2
tmp4 := π_{CN} (tmp3)

- Pipelining is not always possible, e.g., for all operations that include sorting (*blocking operation*).
- Pipelining can be executed in either *demand driven* or *producer driven* fashion.



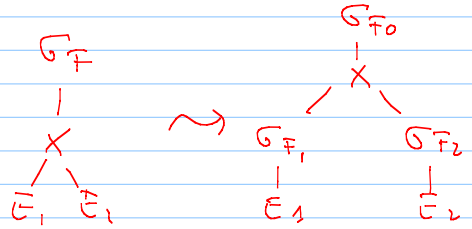
set-minus is non-monotonic

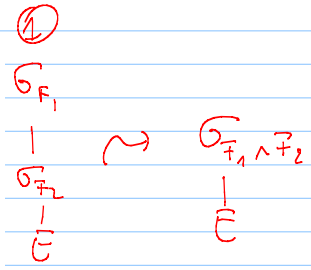


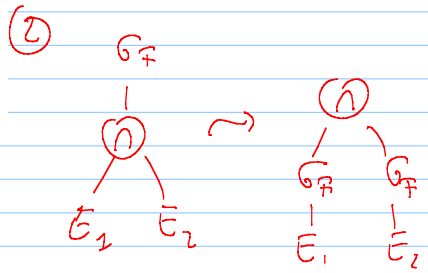
Equivalence Rules (for expressions E, E_1, E_2, conditions F_i)

Applying distribution and commutativity of relational algebra operations

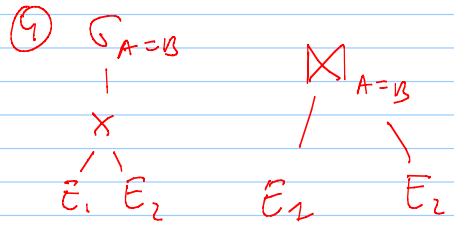
1. $\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$
2. $\sigma_F(E_1\ [\cup, \cap, -]\ E_2) \equiv \sigma_F(E_1)\ [\cup, \cap, -]\ \sigma_F(E_2)$
3. $\sigma_F(E_1 \times E_2) \equiv \sigma_{F0}(\sigma_{F1}(E_1) \times \sigma_{F2}(E_2))$;
   $F \equiv F0 \wedge F1 \wedge F2$, Fi contains only attributes of E_i, $i = 1, 2$.

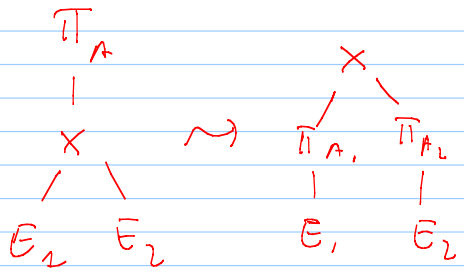






4. $\sigma_{A=B}(E_1 \times E_2) \equiv E_1 \underset{A=B}{\bowtie} E_2$
5. $\pi_A(E_1\ [\cup, \cap, -]\ E_2) \not\equiv \pi_A(E_1)\ [\cup, \cap, -]\ \pi_A(E_2)$
6. $\pi_A(E_1 \times E_2) \equiv \pi_{A1}(E_1) \times \pi_{A2}(E_2)$,
   with $Ai = A \cap \{$ attributes in $E_i\}$, $i = 1, 2$.





7. $E_1 \; [\cup, \cap] \; E_2 \equiv E_2 \; [\cup, \cap] \; E_1$
   $(E_1 \cup E_2) \cup E_3 \equiv E_1 \cup (E_2 \cup E_3)$ (the analogous holds for $\cap$)

$(E_1 \cap E_2) \cap E_3$
$= E_1 \cap (E_2 \cap E_3)$

8. $E_1 \times E_2 \equiv \pi_{A1,A2}(E_2 \times E_1)$
   $(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$
   $(E_1 \times E_2) \times E_3 \equiv \pi((E_1 \times E_3) \times E_2)$

9. $E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$ $\quad (E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$