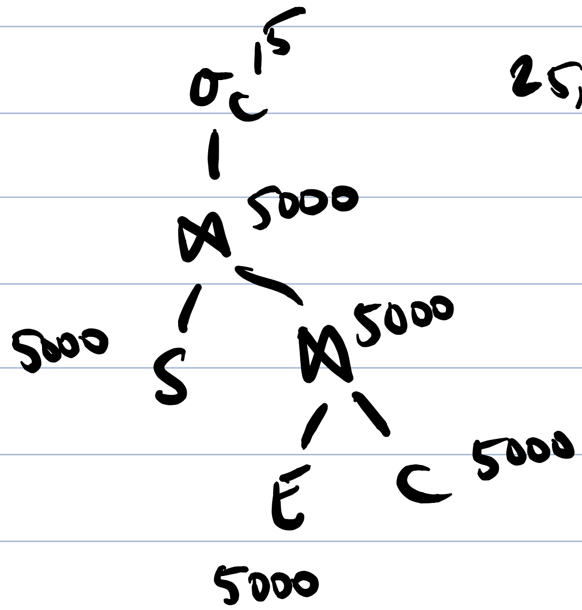
natural join between R and S and S is a B+ tree with depth d_S

$$\text{cost}(R \bowtie S) = \text{cost}(R) + d_S |R|$$

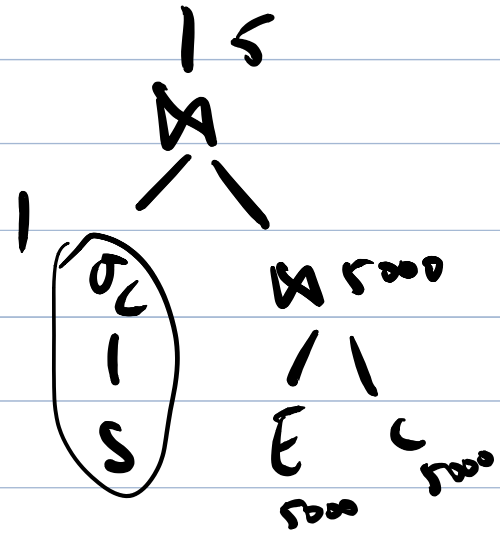
nested loop join

$$\text{cost}(R \bowtie S) = \text{Cost}(R) + \frac{|R|}{b}\,\text{Cost}(S)$$



25,005

vs.

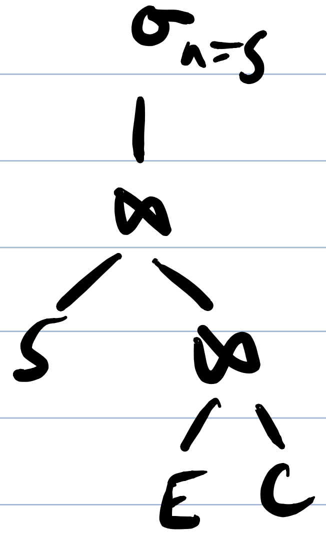


15005

$$R \bowtie S \equiv \pi(S \bowtie R)$$

reorder columns

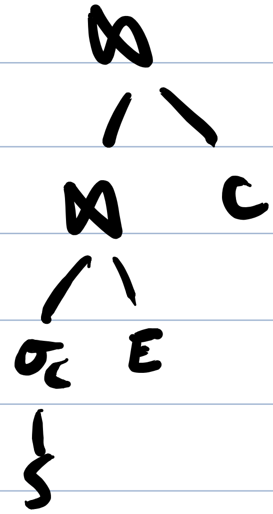
- nested loop join cost is not symmetric e.g. 1) $\sigma_{name='Smith'}(S) \bowtie (E \bowtie C)$



we produce $E \bowtie C$, 1 tuple / course registration by any student

~ 5000 tuples

2)



intermediate rel^n 1 tuple / each course reg. by Smith student

say 10 Smiths

$\Rightarrow$ ~ 50 tuples

- all operators (except sorting) operate without storing intermediate results
  - $\Rightarrow$ iterator protocol in constant storage
  - $\Rightarrow$ no recomputation for <u>left-deep</u> query plans
- general pipelined plans lead to recomputation
  - $\Rightarrow$ we introduce a Store operator
    - ↳ semantically represents the identity

$\text{cost}_c(\text{store}(E)) = \text{cost}_c(E) + \text{cost}_s(E) + \frac{|E|}{b}$

↳ Compute cost

$\text{cost}_s(\text{store}(E)) = \frac{|E|}{b}$

↳ Scanning cost

another approach to improving performance: take advantage of parallelism in hardware
(also parallel, distributed file systems, flash storage)

e.g. Hadoop. Spark (horizontally partition tables)
↳ program through relational algebra

Summary

- RA is the basis for efficient implementation of SQL
- perf. depends on physical schema design
- Compiling insert is also an optimization problem!
- RA allows use of efficient DS/Algos. Bottom-up semantics
- remains: concurrency control + durability

Execution of Transactions

Concurrency control assumptions

1) fix a db: a set of objects R/Wed by transactions

2) a transaction T_i is a sequence of operations concluding with a commit request c_i of T_i.

3) for a set of transactions $\{T_1, \dots, T_k\}$ we want to produce a schedule S of operations s.t. every op $o_i \in T_i$ appears also in S and T_i's operations in S are ordered the same way as in T_i

(transaction comes in as a stream, built incrementally)

goal: produce a correct schedule with maximal parallelism

(3 big db ideas: data independence, quantification, transactions)

Definition (Serializability)

An execution of S is said to be serializable if it is equivalent to a serial execution of the same transactions.

e.g. $T_1 := w_1[x], w_2[y], c_1$

$T_2 := r_2[x], r_2[y], c_2$

interleaved : $S_a = w_1(x), r_2(x), w_1(y), r_2(y)$

equivalent serial exec : $S_b = w_1(x) w_1(y) r_2(x) r_2(y)$

interleaved execution w/ no equivalent serial exec:

$$S_c = w_1[x]\ r_2[x]\ r_2[y]\ w_1[y]$$