>>

Nested-loop Join

∧ >>

❖ Join Example

SQL query on student/enrolment database:

```
select E.subj, S.name
from   Student S join Enrolled E on (S.id = E.stude)
order  by E.subj
```

And its relational algebra equivalent:

Sort[subj] (*Project*[subj,name] (*Join*[id=stude](Student,Enrolled)))

Database: $r_S = 20000$, $c_S = 20$, $b_S = 1000$, $r_E = 80000$, $c_E = 40$, $b_E = 2000$

We are interested only in the cost of *Join*, with N buffers

<< ^ >>

❖ Nested Loop Join

Basic strategy (R.a ⋈ S.b):

```
Result = {}
for each page i in R {
   pageR = getPage(R,i)
   for each page j in S {
      pageS = getPage(S,j)
      for each pair of tuples tR,tS
                       from pageR,pageS {
         if (tR.a == tS.b)
            Result = Result ∪ (tR:tS)
} } }
```

Needs input buffers for R and S, output buffer for "joined" tuples

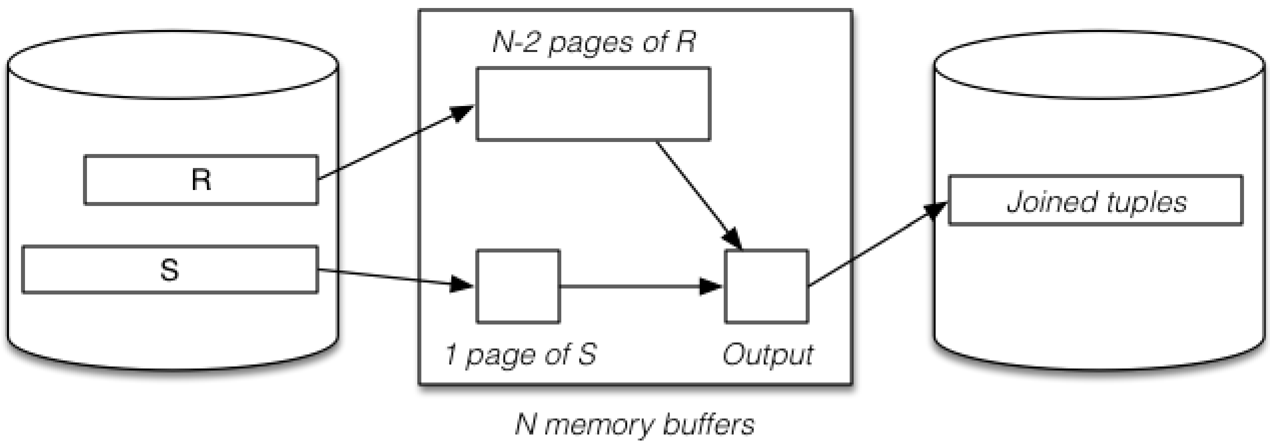
Terminology: R is outer relation, S is inner relation

Cost = $b_R \,.\, b_S$ ... ouch!

❖ Block Nested Loop Join

Method (for *N* memory buffers):

- read *N-2*-page chunk of *R* into memory buffers
- for each *S* page
  check join condition on all `(tR,tS)` pairs in buffers
- repeat for all *N-2*-page chunks of *R*



❖ Block Nested Loop Join (cont)

Best-case scenario: $b_R \leq N\text{-}2$

- read b_R pages of relation R into buffers
- while whole R is buffered, read b_S pages of S

Cost = $b_R + b_S$

Typical-case scenario: $b_R > N\text{-}2$

- read $ceil(b_R/(N\text{-}2))$ chunks of pages from R
- for each chunk, read b_S pages of S

Cost = $b_R + b_S \,.\, ceil(b_R/N\text{-}2)$

Note: always requires $r_R.r_S$ checks of the join condition

<< ∧ >>

❖ Cost on Example Query

With N = 12 buffers, and S as outer and E as inner

- Cost = $b_S + b_E.ceil(b_S/(N\text{-}2))$ = $1000 + 2000.ceil(1000/10)$ = 201000

With N = 12 buffers, and E as outer and S as inner

- Cost = $b_E + b_S.ceil(b_E/(N\text{-}2))$ = $2000 + 1000.ceil(2000/10)$ = 202000

With N = 102 buffers, and S as outer and E as inner

- Cost = $b_S + b_E.ceil(b_S/(N\text{-}2))$ = $1000 + 2000.ceil(1000/100)$ = 21000

With N = 102 buffers, and E as outer and S as inner

- Cost = $b_E + b_S.ceil(b_E/(N\text{-}2))$ = $2000 + 1000.ceil(2000/100)$ = 22000

COMP9315 21T1 ♢ Nested-loop Join ♢ [5/8]

<< ^ >>

❖ Block Nested Loop Join

Why block nested loop join is actually useful in practice ...

Many queries have the form

```
select *
from   R join S on (R.i = S.j)
where  R.x = K
```

This would typically be evaluated as

```
Tmp = Sel[x=K](R)
Res = Join[i=j](Tmp, S)
```

If **`Tmp`** is small ⇒ may fit in memory (in small #buffers)

<< ∧ >>

❖ Index Nested Loop Join

A problem with nested-loop join:

- needs repeated scans of *entire* inner relation *S*

If there is an index on *S*, we can avoid such repeated scanning.

Consider *Join[i=j](R,S)* :

```
for each tuple r in relation R {
    use index to select tuples
        from S where s.j = r.i
    for each selected tuple s from S {
        add (r,s) to result
}   }
```

❖ Index Nested Loop Join (cont)

This method requires:

- one scan of R relation (b_R)
  - only one buffer needed, since we use R tuple-at-a-time
- for each tuple in R (r_R), one index lookup on S
  - cost depends on type of index and number of results
  - best case is when each $R.i$ matches few S tuples

Cost = $b_R + r_R.Sel_S$ (Sel_S is the cost of performing a select on S).

Typical Sel_S = 1-2 (hashing) .. b_q (unclustered index)

Trade-off: $r_R.Sel_S$ vs $b_R.b_S$, where $b_R \ll r_R$ and $Sel_S \ll b_S$

Produced: 28 Mar 2021