CIS 560 – Database System Concepts

Lecture 26

Operator Algorithms

November 4, 2013

Credits for slides: Chang, Ullman, Whitehead.

Outline

- Query execution 15.1-15.6
- Query optimization 16

2

Planning

- Assignment 8 (indexes) due 11/8
- Assignment 9 (query optimization) due 11/15
- Exam 2 (assignments 6-9) – 11/20
- Project assignment due 11/22
- Quiz from special topics – 12/06
- Project presentations 12/9, 12/11, 12/13
- Project reports – finals weeks

3

Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators (operator algorithms), with a focus on join

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Join Physical Operators

Logical operator:
Supplies(sno,pno,price) $\bowtie_{pno=pno}$ **Part(pno,pname,psize,pcolor)**

Three physical operators for the join, assuming the tables are in main memory.

Nested Loop Join
Merge join
Hash join

External Memory Algorithms

- Data is too large to fit in main memory
- Issue: disk access is 3-4 orders of magnitude slower than memory access
- Assumption: runtime dominated by # of disk I/O's; will ignore the main memory part of the runtime

Cost Parameters

In database systems the data is on disk
The *cost* of an operation = total number of I/Os

Cost parameters:

- B(R) = number of blocks for relation R
- T(R) = number of tuples in relation R
- V(R, a) = number of distinct values of attribute *a*
- M = size of main memory buffer pool, in blocks

Facts: (1) B(R) << T(R)
(2) When *a* is a key, V(R,a) = T(R)
When *a* is not a key, V(R,a) << T(R)

Ad-hoc Convention

- We assume that the operator *reads* the data from disk
- We assume that the operator *does not write* the data back to disk (e.g.: pipelining)
  - Need to count it separately when applicable
- Thus:

Any main memory join algorithms for R ⋈ S: Cost = B(R)+B(S)

Any main memory grouping γ(R): Cost = B(R)

Sequential Scan of a Table R

- When R is *clustered*
  - Blocks consists only of records from this table
  - B(R) << T(R)
  - Cost = B(R)
- When R is *unclustered*
  - Its records are placed on blocks with other tables
  - B(R) ≈ T(R)
  - Cost = T(R)

Types of Algorithms

- One-pass algorithms (Sec. 15.2 and 15.3)
- Index-based algorithms (Sec 15.6)
- Two-pass algorithms (Sec 15.4 and 15.5)

Types of Algorithms

- One-pass algorithms (Sec. 15.2 and 15.3)
  - By "one pass", we mean that the operator reads its operands only once. It does not write intermediate results back to disk.
- Index-based algorithms (Sec 15.6)
- Two-pass algorithms (Sec 15.4 and 15.5)

Nested Loop Joins

- Tuple-based nested loop join R ⋈ S

```
for each tuple r in R do
  for each tuple s in S do
    if r and s join then output (r,s)
```

R=outer relation
S=inner relation

- Cost: B(R)+T(R)B(S)
- One-pass only over the outer relation R
  - But S is read many times

Nested-Loop Join

- Block-based nested loop join R ⋈ S

```
for each block of tuples r in R do
   for each block of tuples s in S do
        for each pair of tuples (r,s)
             if r and s join then output(r,s)
```

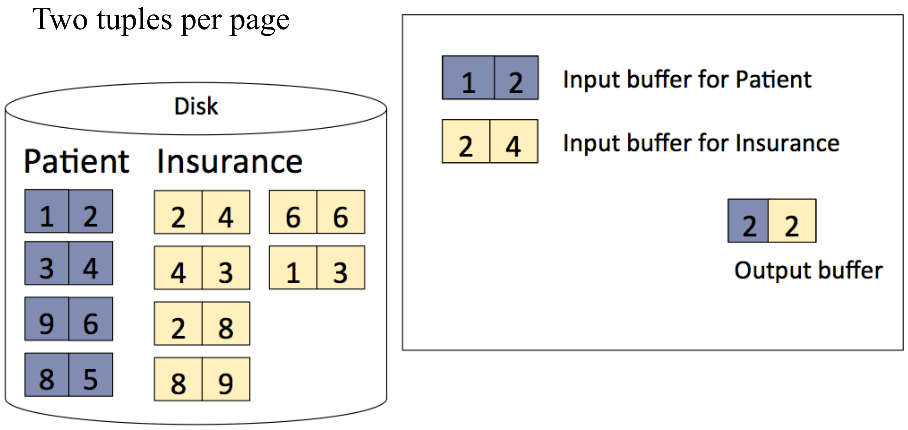
- Cost: B(R)+B(R)B(S)

Nested Loop Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)

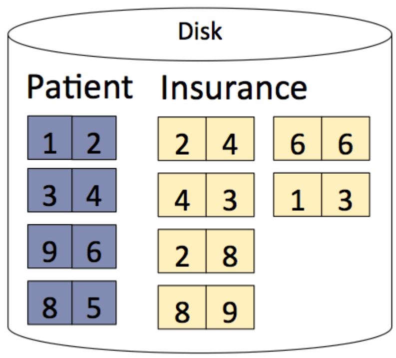
Patient ⋈ Insurance

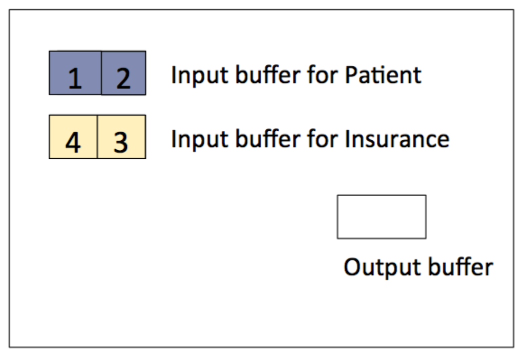
Two tuples per page



Nested Loop Example

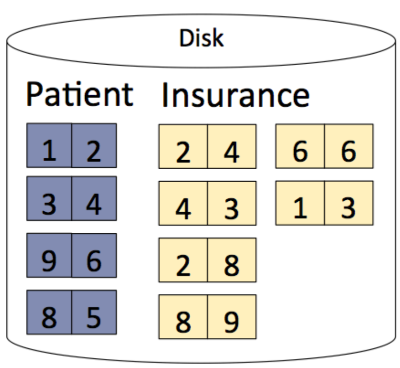
Patient $\bowtie$ Insurance

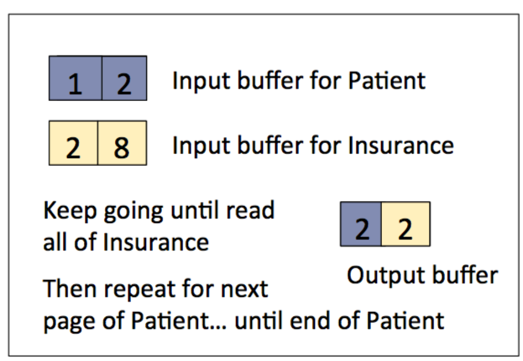




Nested Loop Example

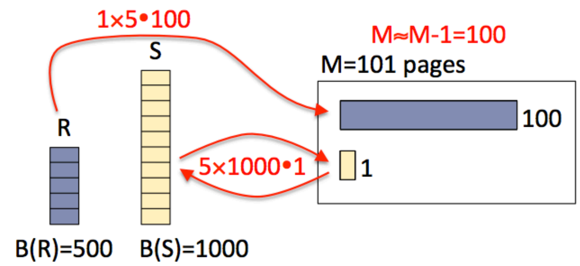
Patient $\bowtie$ Insurance





Cost: B(R) + B(R) B(S)

Block Nested-Loop Join: Example

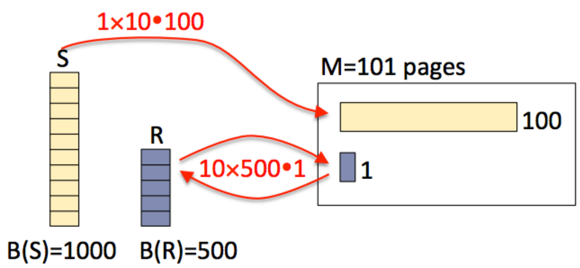


Book example 15.4

Cost R: 500
Cost S: 5000 = 5•1000
SUM: **5500**

B(R) + B(R)/M • B(S)

500 + (500/100)•1000 = **5500**



Cost S: 1000
Cost R: 5000 = 10•500
SUM: **6000**

B(S) + B(S)/M • B(R)

1000 + (1000/100)•500 = **6000**

Hash Join

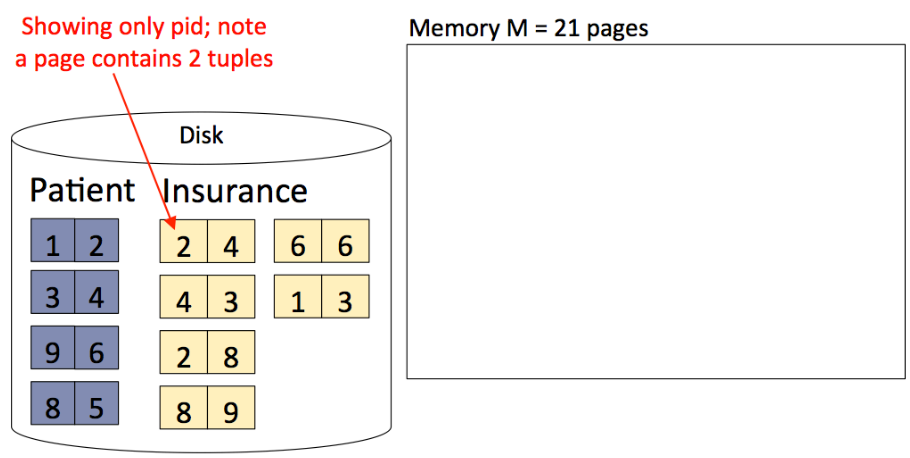
Hash join: $R \bowtie S$

- Scan R, build buckets in main memory
- Then scan S and join
- Cost: B(R) + B(S)

- One-pass algorithm when B(R) ≤ M

Hash Join Example

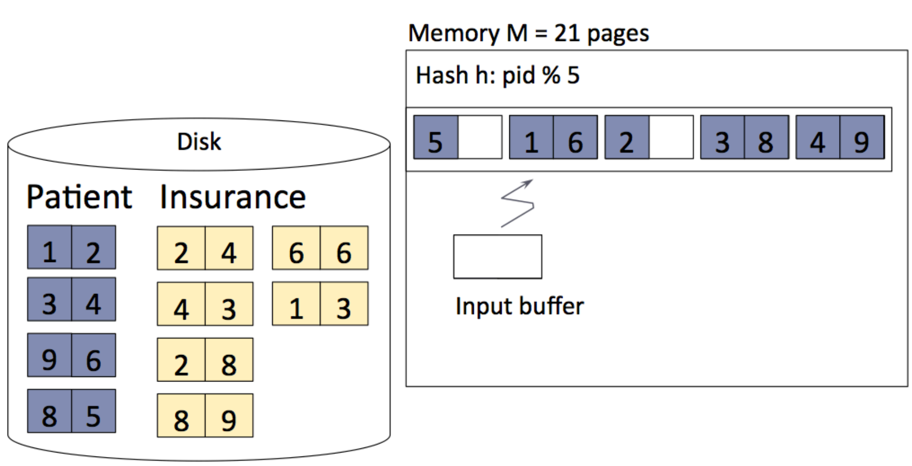
Patient $\bowtie$ Insurance



Hash Join Example

Patient $\bowtie$ Insurance

Step 1: Scan Patient and create hash table in memory

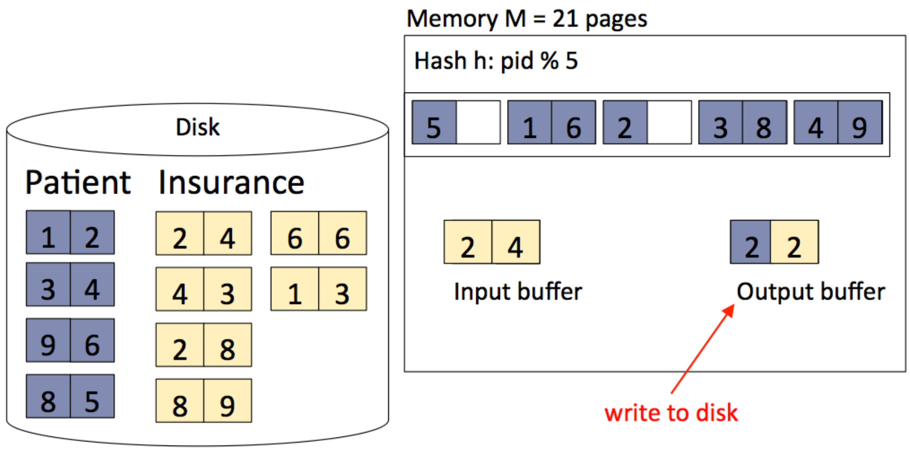


Hash Join Example

Patient $\bowtie$ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

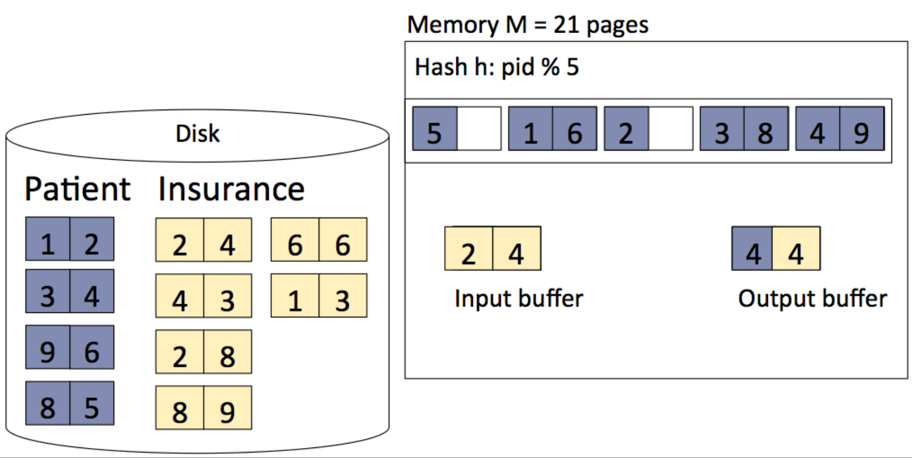


Hash Join Example

Patient $\bowtie$ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

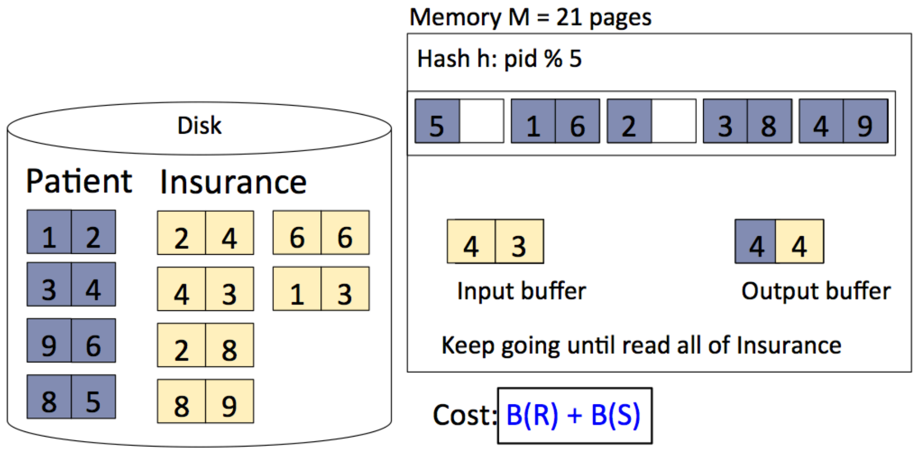


Hash Join Example

Patient ⋈ Insurance

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

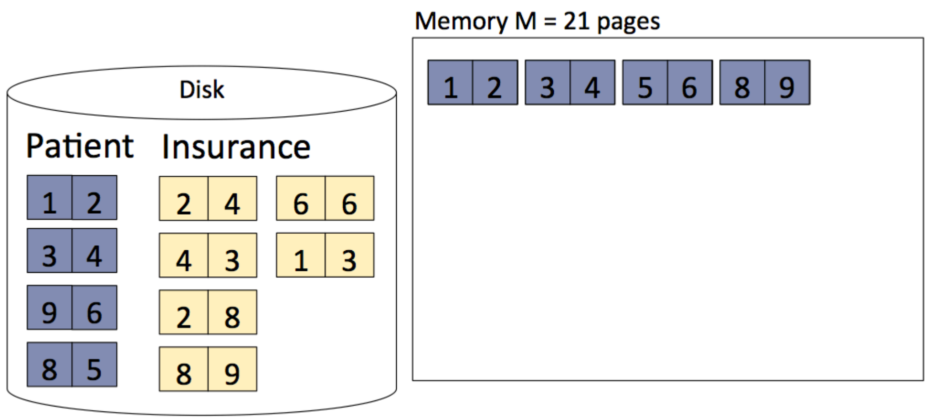


Sort-Merge Join

- Sort-merge join: R ⋈ S
  - Scan R and sort in main memory
  - Scan S and sort in main memory
  - Merge R and S
- Cost: B(R) + B(S)
- One pass algorithm when B(S) + B(R) ≤ M
- Typically, this is NOT a one pass algorithm

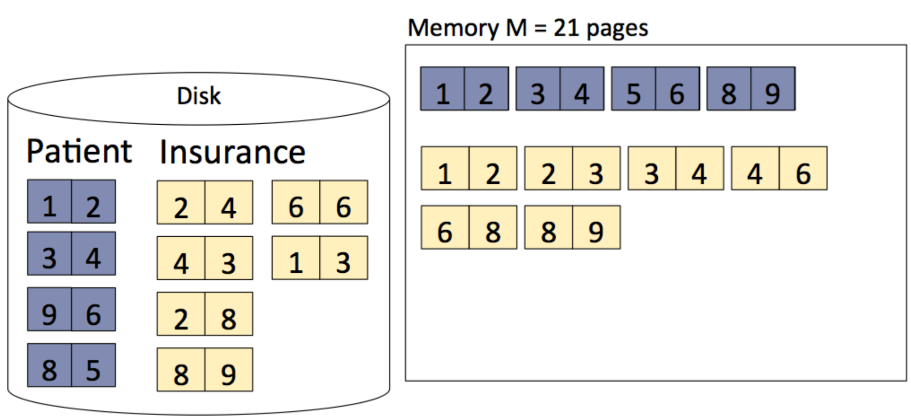
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory



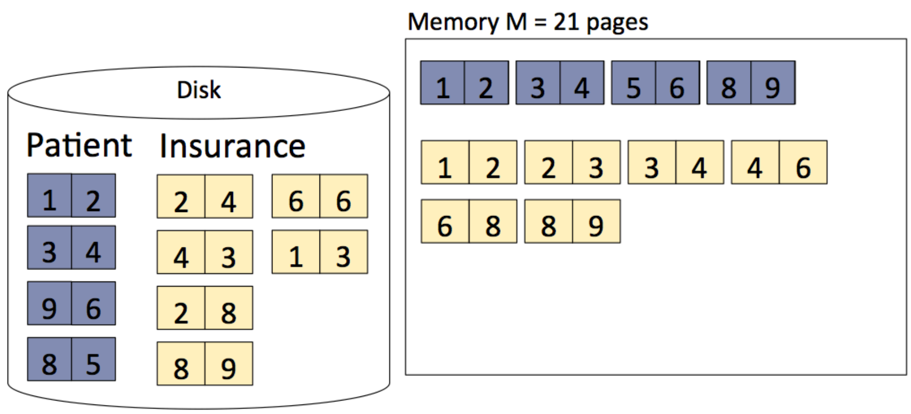
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory



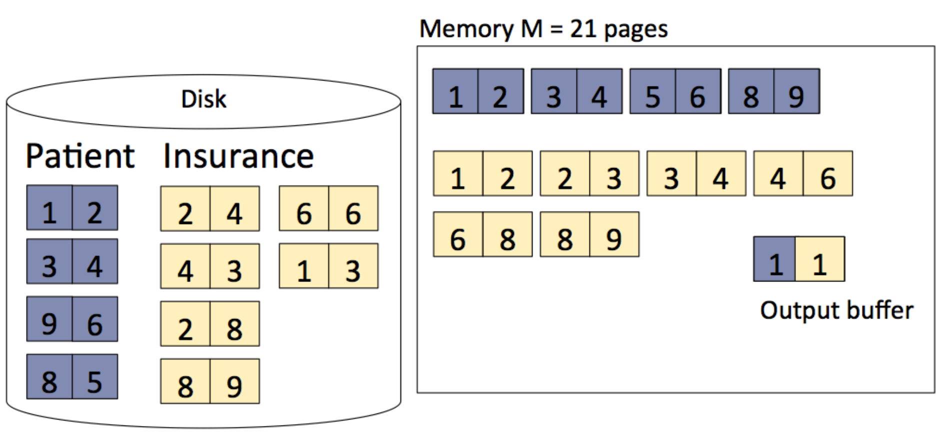
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory
Step 3: Merge Patient and Insurance



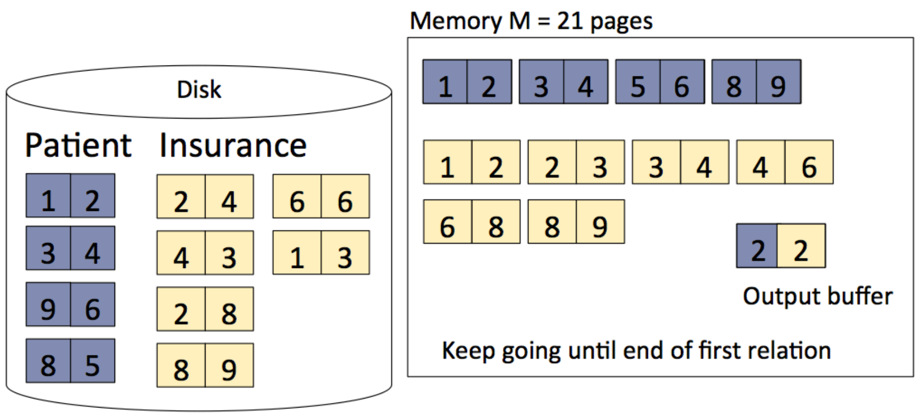
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory
Step 3: Merge Patient and Insurance



Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory
Step 3: Merge Patient and Insurance



Types of Algorithms

- One-pass algorithms (Sec. 15.2 and 15.3)
- Index-based algorithms (Sec 15.6)
- Two-pass algorithms (Sec 15.4 and 15.5)

Access Methods

- Heap file
  - Scan tuples one at the time
- Hash-based index
  - Efficient selection on equality predicates
  - Can also scan data entries in index
- Tree-based index
  - Efficient selection on equality or range predicates
  - Can also scan data entries in index

Index Based Selection

Selection on equality: $\sigma_{a=v}(R)$

- V(R,a) = # of distinct values of attribute a

Expected number of pages

- Cost of clustered index on *a*: B(R)/V(R,a)
- Cost of unclustered index on *a*: T(R)/V(R,a)

Expected number of tuples

- Note: we ignored I/O cost for index pages

Index Based Selection

- Example:

T(R) = 100,000
B(R) = 2,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

- Table scan (assuming R is clustered)?
- Index based selection:
  - If index is clustered?
  - If index is unclustered?

Index Based Selection

- Example:

T(R) = 100,000
B(R) = 2,000
V(R, a) = 20

cost of $\sigma_{a=v}(R)$ = ?

- Table scan:
  - B(R) = 2,000 I/Os
- Index based selection:
  - If index is clustered: B(R)/V(R,a) = 100 I/Os
  - If index is unclustered: T(R)/V(R,a) = 5,000 I/Os

Rule of thumb: don't build unclustered indexes when V(R,a) is small!

Index Based Join

- Index-nested loop join R ⋈ S
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

```
for each tuple r in R do
   lookup the tuple(s) s in S using the index
   output (r,s)
```

Index Based Join

Expected number of blocks from S that join with a tuple from R

Cost:

- If index is clustered: B(R) + T(R)B(S)/V(S,a)
- If unclustered: B(R) + T(R)T(S)/V(S,a)

Expected number of tuples from S that join with a tuple from R