6.830 Lecture 11

Recap

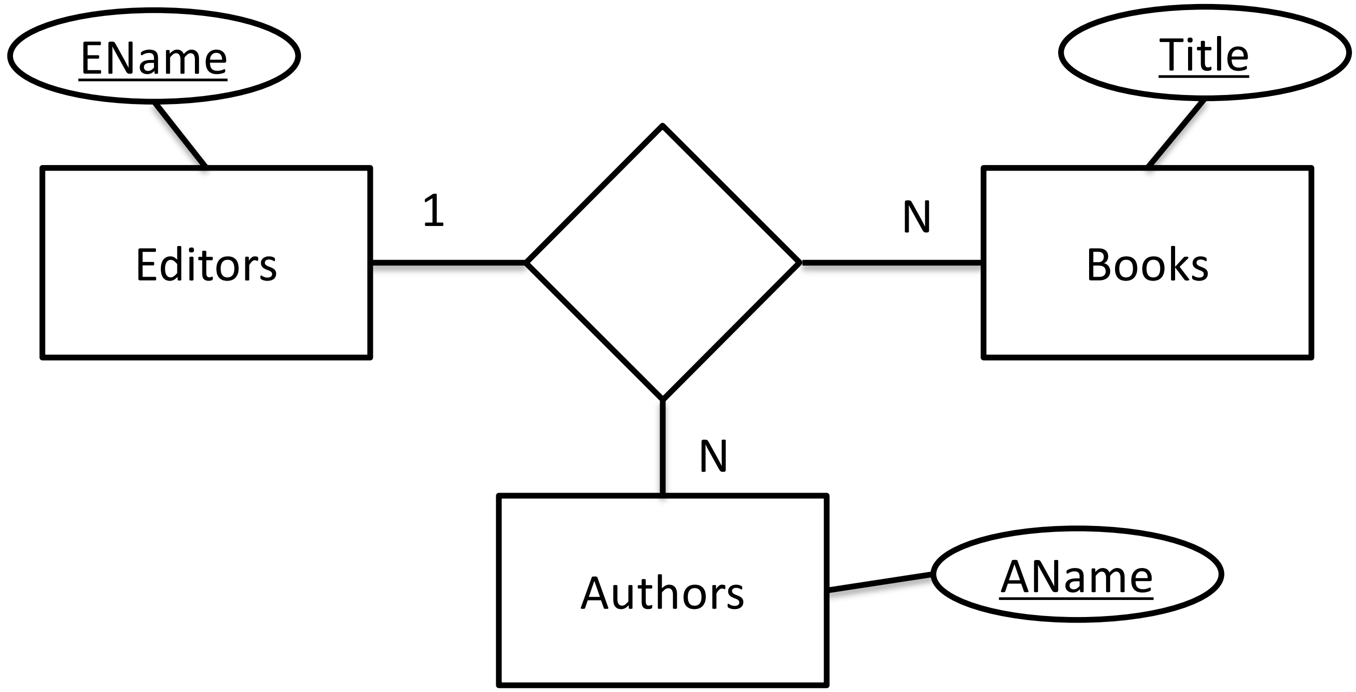
10/15/2018

Celebration of Knowledge

- 1.5h
- No phones, No laptops
- Bring your Student-ID
- The 5 things allowed on your desk
  - Calculator allowed
  - 4 pages (2 pages double sided) of your liking
  - Student-id
  - The exam (will have extra pages for notes)
  - Sugary stuff (i.e., food, drinks to keep you going)

ER Model

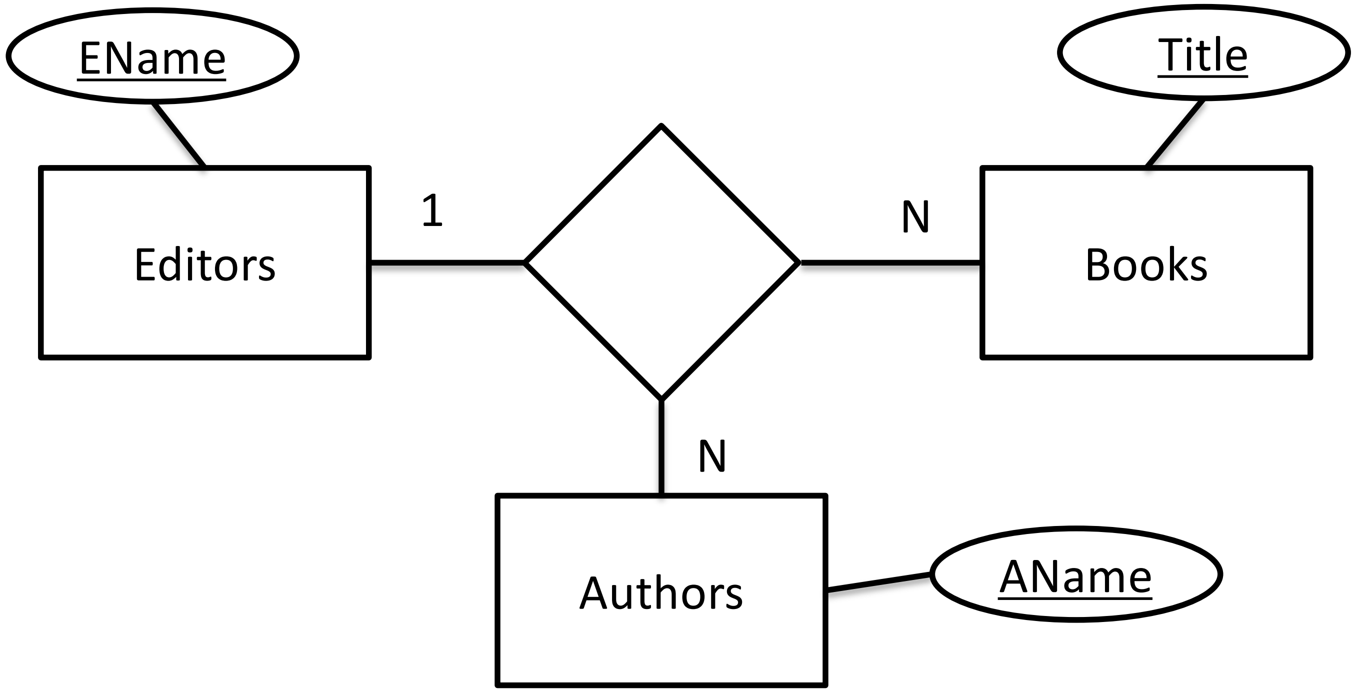
Clicker: ER Model



Suppose that there are 300 authors, 1000 books and 10 editors. What is the maximal number of triples that R contains?

a) 300
b) 3000
c) 10,000
d) 300,000
e) 3,000,000

Clicker: ER Model



Suppose that there are 300 authors, 1000 books and 10 editors. What is the maximal number of triples that R contains?

a) 300
b) 3000
c) 10,000
d) 300,000
e) 3,000,000

Solution: 300 * 1000: 300,000
As every author and book unique identifies the editor

Type of Indexes

Question

What indexes would you create for the following queries (assuming each query is the only query the database runs)

```
SELECT MAX(sal) FROM emp
   B+Tree on emp.sal
SELECT sal FROM emp WHERE id = 1
   Hash index on emp.id
SELECT name FROM emp WHERE sal > 100k
   B+Tree on emp.sal (maybe)
SELECT name FROM emp WHERE sal > 100k AND dept = 2
   B+tree on emp.sal (maybe), Hash on dept.dno (maybe)
```

Clicker

What is the estimated IO cost for

```
SELECT *
FROM T
WHERE T.time = 109808098
```

Assuming that id is unique and is indexed using a non-clustered B-Tree with fanout B and N records

a) 1 b) 2 c) 3 d) LOG_B N e) LOG_B N + 1

Clicker - Solution

You can argue that probably most of those answers are correct:

a) If you assume the index is in memory and you only have a random IO for the data

b & c) Most B-Trees in practice are only 2-3 levels deep with any reasonable fanout. Assuming that the root-note is in memory (a very reasonable assumption), it requires 1-2 random IOs to traverse the remaining memory, plus one random IO to get to the data page.

d) This is the cost to traverse the B-Tree. However, if you assume that the root note is in memory as before, this is a good general estimate

e) Is the general case and implicitely assume even the root node is not in memory.

Important: It depends on the assumptions and you should always state them.

For Query Optimizers Constants matter a lot (O-Notation is not enough).

Joins

Clicker: Index Nested Loop

```
for s in S
   find matches in R
```

What is the expect IO cost assuming a non-clustered hash-index?

```
a) |S| + {S}
b) |S| + {R}
c) |S| + {S} * 2
```

```
|S| = NB of pages, {S} = NB of records
```

Clicker: Solution

```
for s in S
   find matches in R
```

What is the expect IO cost assuming a non-clustered hash-index?

a) |S| + {S} The most common (simplified) cost estimate

b) |S| + {R} Usually better if {R} > {S}

c) |S| + {S} * 2 Considers the indirection for a non-clustered index.

Question: What about |S| + {R} * 2. When is that the bette estimate?

Clicker Question

- Assuming disk can do 100 MB/sec I/O, and 10ms / seek
- And the following schema:

```
grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))
```

1. Estimate time to sequentially scan grades, assuming it contains 1M records (Consider: field sizes, headers)

2. Estimate time to join these two tables, using nested loops, assuming students fits in memory but grades does not, and students contains 10K records.

Clicker Question

- Assuming disk can do 100 MB/sec I/O, and 10ms / seek
- And the following schema:

```
grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))
```

1. Estimate time to sequentially scan grades, assuming it contains 1M records (Consider: field sizes, headers)

(a) .23 sec   (b) 0.251 sec   (c) .21 sec   (d) 0.5 sec

Int = 8 Bytes, 1MB = 1000KB

Seq Scan Grades

```
grades (cid int, g_sid int, grade char(2))
Record-Size:
8 bytes (cid) + 8 bytes (g_sid) + 2 bytes
(grade) + 4 bytes (header) = 22 bytes
```

Data-Size: **22 MB**

Scan-Time: 22 MB / 100 MB/sec + seek-time

= .22 sec + 0.01 sec → **.23 sec**

Clicker Question

Assuming disk can do 100 MB/sec I/O, and 10ms / seek, and the following schema:

```
grades (cid int, g_sid int, grade char(2))
students (s_int, name char(100))
```

Estimate time to join these two tables, using nested loops, assuming students fits in memory but grades does not, and students contains 10K records.

(a) .241 sec (b) 0.251 sec (c) .211 sec (d) 0.502 sec

Int = 8 Bytes, 1MB = 1000KB

Grace Hash

Algorithm:

Partition:

Suppose we have P partitions, and H(x) → [0…P-1]

Choose P = |S| / M ➔ P ≤ sqrt(|S|) //may need to leave a little slop for imperfect hashing

Allocate P 1-page output buffers, and P output files for R

For each r in R:

  Write r into buffer H(r)

  If buffer full, append to file H(r)

Allocate P output files for S

For each s in S:

  Write s into buffer H(s)

  if buffer full, append to file H(s)

Join:

For i in [0,…,P-1]

  Read file i of R, build hash table

  Scan file i of S, probing into hash table and outputting matches

Need one page of RAM for each of P partitions

Since
M > sqrt(|S|) and
P ≤ sqrt(|S|), all is well

Total I/O cost: Read |R| and |S| twice, write once

3(|R| + |S|) I/Os

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2

P output buffers

F0	F1	F2

P output files

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
		5

F0	F1	F2

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
	4	5

F0	F1	F2

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
3	4	5

F0	F1	F2

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
3	4	5
6		

F0	F1	F2

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
3	4	5
6		

Need to flush R0 to F0!

F0	F1	F2

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
	4	5

F0	F1	F2
3		
6		

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	4	5

F0	F1	F2
3		
6		

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	4	5
		14

F0	F1	F2
3		
6		

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	4	5
	1	14

F0	F1	F2
3		
6		

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	4	5
	1	14

F0	F1	F2
3		
6		

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9		5
		14

F0	F1	F2
3	4	
6	1	

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	7	5
		14

F0	F1	F2
3	4	
6	1	

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	7	5
		14

F0	F1	F2
3	4	
6	1	

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	7	

F0	F1	F2
3	4	5
6	1	14

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2
9	7	11

F0	F1	F2
3	4	5
6	1	14

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R0	R1	R2

F0	F1	F2
3	4	5
6	1	14
9	7	11

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Example

P = 3; H(x) = x mod P

Matches:

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Example

P = 3; H(x) = x mod P

Matches:

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

Matches:
3,3

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

Matches:
3,3

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

Matches:
3,3
9,9

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

Matches:
3,3
9,9

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

Matches:
3,3
9,9
6,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

Load F0 from R into memory

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Scan F0 from S

Example

P = 3; H(x) = x mod P

Matches:
3,3
9,9
6,6

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Example

$P = 3; H(x) = x \bmod P$

Matches:
3,3
9,9
6,6
7,7
4,4

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Example

P = 3; H(x) = x mod P

Matches:
3,3
9,9
6,6
7,7
4,4

R=5,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R Files

F0	F1	F2
3	4	5
6	1	14
9	7	11

S Files

F0	F1	F2
3	7	2
12	4	8
9		
15		
6		

Sort-Merge Join
(|R| + |S| < M -- in memory)

```
Sort R, Sort S
Merge
```

```
|R| + |S| i/os
```

Join Processing in Database Systems with Large Main Memories

LEONARD D. SHAPIRO
North Dakota State University

Sort Merge Join

Equi-join of two tables S & R

|S| = Pages in S; {S} = Tuples in S

|S| ≥ |R|

M pages of memory; M > sqrt(|S|)

Algorithm:

- Partition S and R into memory sized sorted runs, write out to disk
- Merge all runs simultaneously

Total I/O cost: Read |R| and |S| twice, write once

3(|R| + |S|) I/Os

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

If each run is M pages and M > sqrt(|S|), then there are at most

$$|S|/\text{sqrt}(|S|) = \text{sqrt}(|S|)$$

runs of S

So if |R| = |S|, we actually need M to be 2 x sqrt(|S|)

[handwavy argument in paper for why it's only sqrt(|S|)]

R1 =

S1 =

R					
1					
3					
4	14	11	7	12	15

UTPUT

Need enough memory to keep 1 page of each run in memory at a time

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6	1	2	8	4
3	9	7	3	9	6
4	14	11	7	12	15

OUTPUT

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4   R2 = 6,9,14   R3 = 1,7,11

S1 = 2,3,7   S2 = 8,9,12   S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6	1	2	8	4
3	9	7	3	9	6
4	14	11	7	12	15

OUTPUT

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6 ⇐	1	2	8 ⇐	4 ⇐
3 ⇐	9	7 ⇐	3 ⇐	9	6
4	14	11	7	12	15

OUTPUT
(3,3)

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6	1	2	8	4
3	9	7	3	9	6
4	14	11	7	12	15

OUTPUT
(3,3)
(4,4)

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6	1	2	8	4
3	9	7	3	9	6
4	14	11	7	12	15

OUTPUT
(3,3)
(4,4)

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6	1	2	8	4
3	9	7	3	9	6
4	14	11	7	12	15

OUTPUT
(3,3)
(4,4)
(6,6)

Example

R=1,4,3,6,9,14,1,7,11

S=2,3,7,12,9,8,4,15,6

R1 = 1,3,4 R2 = 6,9,14 R3 = 1,7,11

S1 = 2,3,7 S2 = 8,9,12 S3 = 4,6,15

R1	R2	R3	S1	S2	S3
1	6 ⇐	1	2	8 ⇐	4
3	9	7 ⇐	3	9	6 ⇐
4 ⇐	14	11	7 ⇐	12	15

OUTPUT
(3,3)
(4,4)
(6,6)
(7,7)

…

Output in sorted order!

Summary

Notation: P partitions / passes over data; assuming hash is O(1)

Sort-Merge	Grace Hash
I/O: 3 (|R| + |S|) CPU: O(P x {S}/P log {S}/P)	I/O: 3 (|R| + |S|) CPU: O({R} + {S})

Summary

Notation: P partitions / passes over data; assuming hash is O(1)

Sort-Merge	Grace Hash
I/O: $3\ (\lvert R\rvert + \lvert S\rvert)$ CPU: $O(P \times \{S\}/P \log \{S\}/P)$	I/O: $3\ (\lvert R\rvert + \lvert S\rvert)$ CPU: $O(\{R\} + \{S\})$

Grace hash is generally a safe bet, unless memory is close to size of tables

Extra cost of sorting makes sort merge unattractive unless there is a way to access tables in sorted order (e.g., a clustered index), or a need to output data in sorted order (e.g., for a subsequent ORDER BY)

Clicker

Assume a Star-Schema:

```
fact(o_id,p_id,zip,discount,amount,)
product(p_id,price,description)
location(zip,city,s_id)
```

And the following query:

```
SELECT *
FROM fact f , product p, location l
WHERE f.zip = l.zip AND f.p_id = p.p_id
AND city=Boston
```

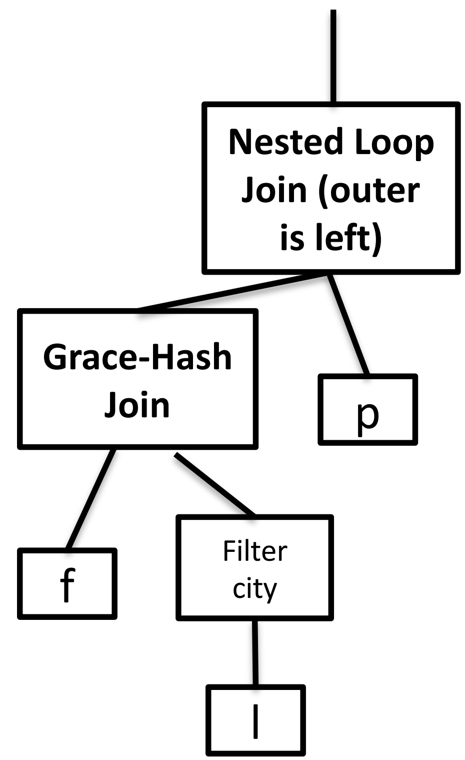
Fact contains 10M, Product contains 10k, Location contains 500

1k records fit into main memory

Would a reasonable optimizer pick the plan on the right?

a) Yes (and why)

b) No (and why not)



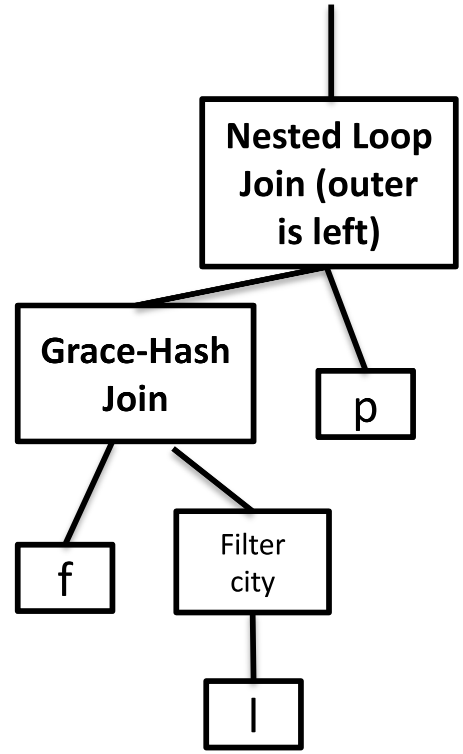
Clicker

Assume a Star-Schema:

```
fact(o_id,p_id,zip,discount,amount,)
product(p_id,price,description)
location(zip,city,s_id)
```

And the following query:

```
SELECT *
FROM fact f , product p, location l
WHERE f.zip = l.zip AND f.p_id = p.p_id
AND city=Boston
```



Fact contains 10M, Product contains 10k, Location contains 500

1k records fit into main memory

Solution: No, you wouldn't use a grace hash join; after the city-filter it would fit into main memory – Hash Join or Nested Loop Join are better options. Furthermore, you would not use a nested loop join for p, as products don't fit into main memory