COSC 460 Lecture 9: SQL (Aggregations) And wrapping up indexing

Professor Michael Hay
Fall 2018

Records for extendible hash index example

Record id	Search Key k	hash(k)	hash(k) *in binary*
r_1	“Alice”	13	“1101”
r_2	“Barry”	6	“0110”
r_3	“Ciao”	7	“0111”
r_4	“Divesh”	7	“0111”
r_5	“Earl”	3	“0011”
r_6	“Ferdinando”	8	“1000”
r_7	“George”	4	“0100”

Extendible hashing: insert

- Compute hash, insert, check for overflow
- Handling overflow:
  - If local depth = global depth, double directory
  - Else (local depth < global depth), split bucket and increase local depth and update directory
  - Rehash entries in overflowing bucket
- *Overflow can still happen*! Example: if many records with ***same*** search key value inserted, directory doubling doesn't help!

Extendible hashing: delete

- Exact “reverse” of insert
- If bucket empty
  - Merge with “sibling”
  - Decrement local depth
  - Update relevant entries in directory
- If local depth < global depth for *all* buckets, then cut directory in half

pollev.com/cosc460

Correct answer: E

Extendible hashing

In the example on the board, many of the inserts caused the directory to double. This is unrealistic. Under what conditions can a record be inserted *without doubling the directory?*

A. The record is inserted into a bucket that is under capacity.

B. The record is inserted into a full bucket but the local depth of the bucket is less than the global depth of the directory.

C. The record is inserted into a full bucket but that bucket is pointed to by more than one virtual bucket in the directory.

D. There is at least one bucket that is under capacity.

E. More than one of the above

Instructions: *I will give you 1-2 minutes to think on your own.*
Vote 1.
Then you will discuss w/ neighbor (1 min).
Vote 2.
Then we'll discuss as class.

	Heap File	Sorted File	Unclustered B+Tree	Clustered B+Tree	Unclustered Hash
Equality search	B	log_2B	log_FB^*	log_FB	1 + 1
Range search	B	log_2B + #matching pages	log_FB^* + #matching records	log_FB + #matching pages	B
Insert	2	log_2B + B	log_FB^* + 1 + 2 (write leaf) + (update heap)	log_FB + 1 (write leaf)	2 + 2 (update index) + (update heap)
Delete	2	Same as insert	Same as insert	Same as insert	Same as insert

- F is “fanout” (# of index entries per page)
- B* number pages needed to store data entries (assuming alternative 2). Typically B* is much less than B!
- Insert cost on tree/hash does *not* count the cost of tree node splits or directory doubling + bucket rehashing (these are rare events)

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

cName	state	enrollment
Colgate	NY	2700
Bucknell	PA	3650
Williams	MA	2000
Cornell	NY	21000

- Write a query to compute total college enrollments by state. On the input relation above, it would produce this output:

state	totalEnroll
MA	2000
NY	23700
PA	3650

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

We want a query that returns for each student, the student's name along with the number of colleges that student applied to. Does this query return the correct result? If not, where is the error?

A. yes, it's correct

B. error in select statement

C. error in from clause

D. error in where clause

E. error in group by clause

F. more than one error

```
select sName, count(*)
from Student S, Apply A
where S.sID = A.sID
group by sName
```

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

cName	state	enrollment
Colgate	NY	2700
Bucknell	PA	3650
Williams	MA	2000
Cornell	NY	21000

- Write a query to find states whose total college enrollment exceeds 20,000. On the input relation above, it would produce this output:

state
NY

Question

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

Suppose the Student and Apply relations were as shown on the right. Consider the following query. Which student is *not* included in the result?

A. Amy

B. Bob

C. Craig

D. Doris

E. More than one is not included

```
select S.sID, sName, count(*)
from Student S, Apply A
where S.sID = A.sID
group by S.sID, sName
having count(*) < 4;
```

Student

```
123,Amy,3.9,1000
234,Bob,3.6,1500
345,Craig,3.5,500
456,Doris,3.9,1000
```

Apply

```
123,Colgate,CS,Y
123,Colgate,english,N
123,Bucknell,CS,Y
123,Cornell,english,Y
234,Bucknell,biology,N
345,Williams,chemistry,Y
```