Updates on B+-Trees: Deletion

- Find the record, delete it.
- Remove the corresponding (search-key, pointer) pair from a leaf node
  - Note that there might be another tuple with the same search-key
  - In that case, this is not needed
- Issue:
  - The leaf node now may contain too few entries
    - Why do we care?
    - Solutions:
      1. If possible, merge with sibling, else
      2. Borrow an entry from a sibling
  - May end up merging all the way to the root
  - In fact, may reduce the height of the tree by one
Delete

**Example of B⁺-Tree Deletion**

Before and after deleting “Srinivasan”

- Deleting “Srinivasan” causes **merging** of under-full leaves
- Parent of merged nodes left w/ only 1 ptr. Can’t merge w/ Cal/Ein/Gol, so **borrows** Gold.
  - Key ‘Gold’ rotates up, ‘Mozart’ rotates down.
Example of B+-Tree Deletion (cont.)

Before and after deletion of “Gold”

- Node with Gold and Katz became underfull, and was merged with its sibling
- Parent node becomes underfull, and is merged with its sibling
  - Value separating two nodes (at the parent) is pulled down when merging
- Root node then has only one child, and is deleted

Another B+Tree Insertion Example

INITIAL TREE

Next slides show the insertion of (125) into this tree
According to the Algorithm in Figure 12.13, Page 495
Another Example: INSERT (125)

Step 1: Split L to create L'

Insert the lowest value in L' (130) upward into the parent P

Another Example: INSERT (125)

Step 2: Insert (130) into P by creating a temp node T
Another Example: INSERT (125)

**Step 3: Create P’; distribute from T into P and P’**

New P has only 1 key, but two pointers so it is OKAY.
This follows the last 4 lines of Figure 12.13 (note that “n” = 4)
\(K'' = 130\). Insert upward into the root

Another Example: INSERT (125)

**Step 4: Insert (130) into the parent (R); create R’**

Once again following the insert_in_parent() procedure, \(K'' = 1000\)
Another Example: INSERT (125)

Step 5: Create a new root

B⁺ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 67
- Typical capacities:
  - Height 3: $67^3 = 300763$ entries
  - Height 4: $67^4 = 20,151,121$ entries
- Can often hold top levels in buffer pool:
  - Level 1 = 67 page = 8 Kbytes
  - Level 2 = 4489 pages = 37 Mbyte
  - Level 3 = 300,000 pages = 2.4 GBytes
B+ Trees: Summary

- Searching:
  - $\log_n(N)$ – Where $n$ is the order, and $N$ is the number of entries

- Insertion:
  - Find the leaf to insert into
  - If full, split the node, and adjust index accordingly
  - Similar cost as searching

- Deletion
  - Find the leaf node
  - Delete
  - May not remain half-full; must adjust the index accordingly

Exam 1 Prep

- What are the hard topics discussed so far?

**Textbook (7th edition):**

- Ch 3 - 5 for SQL
- Ch 6.1- 6.7 for the ER model.
- Ch 2.6 for the relational algebra
- Ch 7.1 - 7.5 for functional dependency theory and it’s uses.
Q5:4.3

Candidate keys: $A, C$

$\sigma$ is extraneous in $\alpha$ iff:
$F \rightarrow F'$, or
$(\alpha - \sigma)^* \text{ includes } \beta$ under $F$

$\sigma$ is extraneous in $\beta$ iff:
$F' \rightarrow F$, or
$\alpha^* \text{ includes } \sigma$ in $F'$

Please make it easier on the graders and use the algorithm in Figure 8.9

$A \rightarrow CD, C \rightarrow ABE, BC \rightarrow A, AE \rightarrow B$
$A \rightarrow CD, C \rightarrow AE, BC \rightarrow A, AE \rightarrow B$ (B extra)
$A \rightarrow CD, C \rightarrow AE, BC \rightarrow A, A \rightarrow B$ (E extra)
$A \rightarrow BCD, C \rightarrow AE, BC \rightarrow A$ (merge)
$A \rightarrow BCD, C \rightarrow AE, C \rightarrow A$ (B extra)
$A \rightarrow BCD, C \rightarrow AE$ (merge)

or

$A \rightarrow CD, C \rightarrow ABE, BC \rightarrow A, AE \rightarrow B$
$A \rightarrow CD, C \rightarrow ABE, BC \rightarrow A, A \rightarrow B$ (E extra)
$A \rightarrow BCD, C \rightarrow ABE, BC \rightarrow A$ (union)
$A \rightarrow BCD, C \rightarrow ABE, C \rightarrow A$ (B extra)
$A \rightarrow BCD, C \rightarrow ABE$ (merge)
$A \rightarrow CD, C \rightarrow ABE$ (B extra)

Q5:

1 Point

Consider a following additional rule for working with FDs:

**Composition:** If $\alpha \rightarrow \beta$ and $\gamma \rightarrow \phi$, then $\alpha \gamma \rightarrow \beta \phi$

Show how this rule can be derived using the basic Armstrong's Axioms (reflexivity, transitivity, and augmentation).

**EXPLANATION**

Augment $\alpha \rightarrow \beta$ to get $\alpha \gamma \rightarrow \beta \gamma$

Augment $\gamma \rightarrow \phi$ to get $\beta \gamma \rightarrow \beta \phi$

Trans: $\alpha \gamma \rightarrow \beta \gamma \rightarrow \beta \phi$
Q5:

**Q6**
1 Point

One way to show that a set of FD's, F, does not imply another FD, x (through Armstrong's Axioms) is to construct a relation instance such that F holds on it, but x doesn't. Construct such a counterexample to show that, on the relation schema: R(A, B, C, D, E), F = {BC → E, A → D, B → D} does not imply functional dependency E → C. You don't need more than two tuples in this case.

EXPLANATION
(1,1,1,1)
(1,1,2,1,1)

Q5:

Consider a relation R(A, B, C, D, E), and the following FDs on it:

B → DE
D → AC
AE → C

**Q8**
1 Point

For the above schema, which of the following is NOT a lossless decomposition?

- R1(A, B, C), R2(B, C, D, E)
- R1(A, C, D), R2(B, D, E)
- R1(A, C, D), R2(A, C, B, E)
- R1(A, B, D, E), R2(B, C)
Exam 1

One reasonable exam would be

1. SQL and relational algebra
   • write an SQL expression
   • write the corresponding relational algebra
2. ER diagram
   • mark cardinality, participation, etc.
   • convert to relations, minimizing where possible.
3. more relational algebra
   • I give relations, expression, you evaluate them
4. I give you a relation schema, FDs
   • you list candidate keys, specify whether in BCNF or 3NF, converting into if so.
5. Functional dependency theory
   • I give you FDs, you produce canonical cover
   • I give you another FD, and you prove correct through use of Armstrong’s Axioms