Final Exam Practice Problems

CSC 321/621 – 5/1/2012

Views

- What is a materialized view?

Views

- What is a materialized view?

A materialized view is a temporary table that is stored in the database to represent a view, which is maintained as the base table(s) are updated.

Views

- Suggest a scenario when materialized view is likely a more appropriate approach than view resolution?

Views

- Suggest a scenario when materialized view is likely a more appropriate approach than view resolution?

- When the view is used heavily, instead of re-generating all or components of the view as done in view resolution, just generate it once and select from the temporary table.

Views

- Given this view definition,

CREATE VIEW HotelBookingcount(hotelNo, bookingCount) AS SELECT h.hotelNo, COUNT(*) FROM Hotel h, Room r, Booking b WHERE h.hotelNo=r.hotelNo AND r.roomNo=b.roomNo GROUP BY h.hotelNo.

How would the following query be implemented using view resolution?

SELECT hotelno FROM HotelBookingCount where hotelNo= 'H001'

Views

Given CREATE VIEW HotelBookingcount(hotelNo, bookingCount) AS SELECT h.hotelNo, COUNT(*) FROM Hotel h, Room r, Booking b WHERE h.hotelNo=r.hotelNo AND r.roomNo=b.roomNo GROUP BY h.hotelNo.

and SELECT hotelno FROM HotelBookingCount where hotelNo= 'H001'

SELECT h.hotelNo FROM Hotel h, Room r, Booking b WHERE h.hotelNo=r.hotelNo AND r.roomNo=b.roomNo AND h.hotelNo = 'H001' GROUP BY h.hotelNo

Indexes

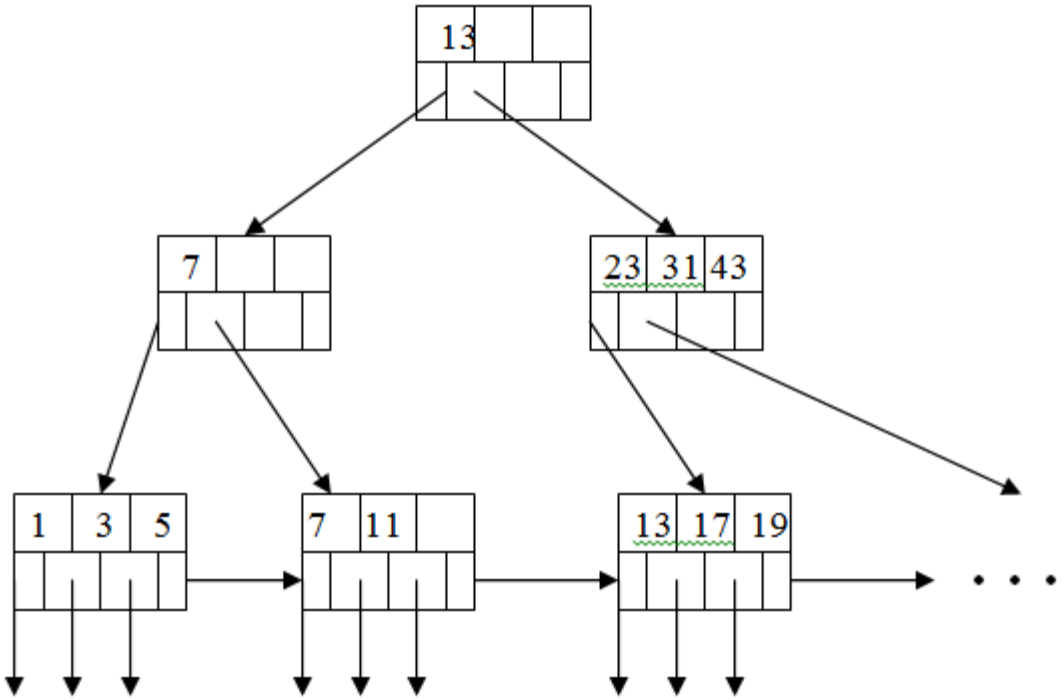
- A) For hashing and B+-tree indices, what are the effective costs of using an index to retrieve a piece of data?
- B) Assume that a change is made to a tuple. What are the costs one may incur in maintaining a B+-tree index? (Note this is an update, not an add)

Indexes

- A) For hashing and B+-tree indices, what are the effective costs of using an index to retrieve a piece of data?
- B) Assume that a change is made to a tuple. What are the costs one may incur in maintaining a B+-tree index? (Note this is an update, not an add)

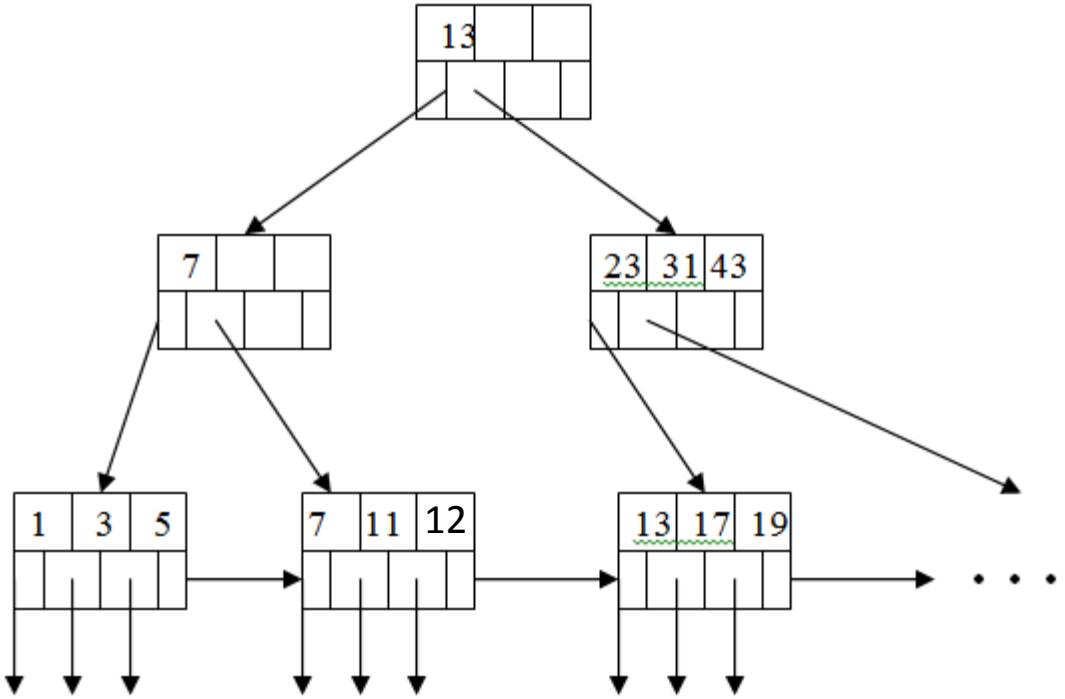
- *Indexes provide quick access to data, usually on the order of O(c), where c is some small constant (hashing: O(1) effectively, B+-tree: O(height), which is small)*
- *A change to a relation on a non-indexed field requires no changes to the index. A change to a relation on the indexed field may require deletion of a pointer and insertion at another point in the index. This insertion at another point may require splits up through the nodes if the nodes were full.*

Indexes



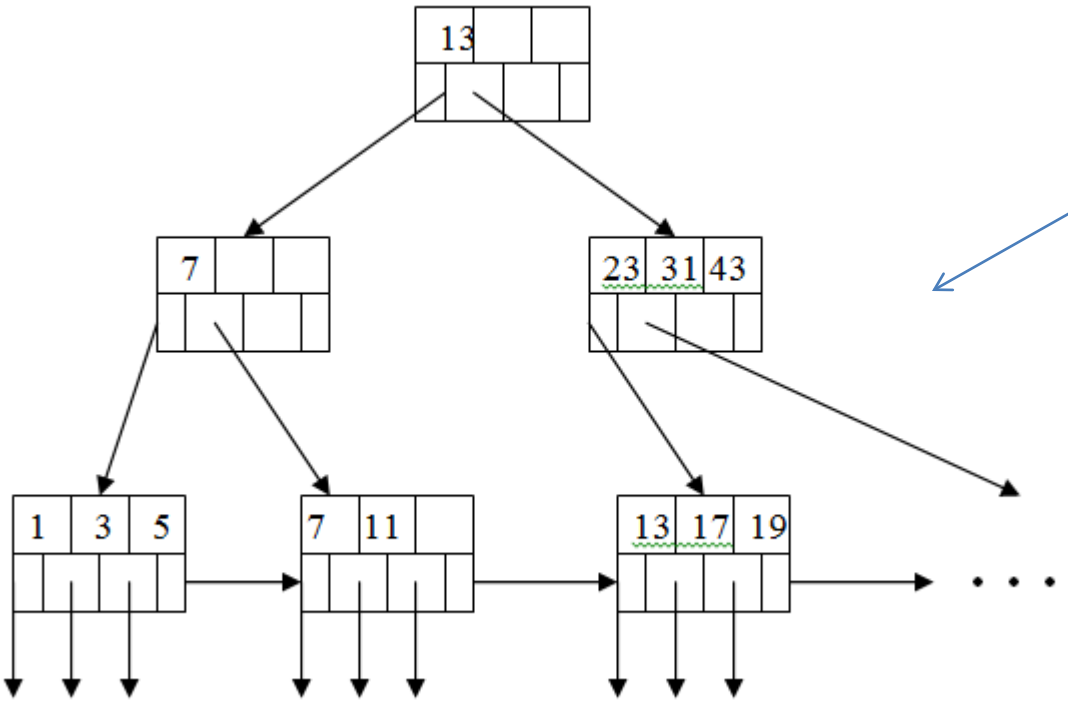
- Given the B+-tree above, show the insertion of value 12 and describe how the process works to get to the insertion point.
- Pretend you didn't insert 12 but instead inserted 15.

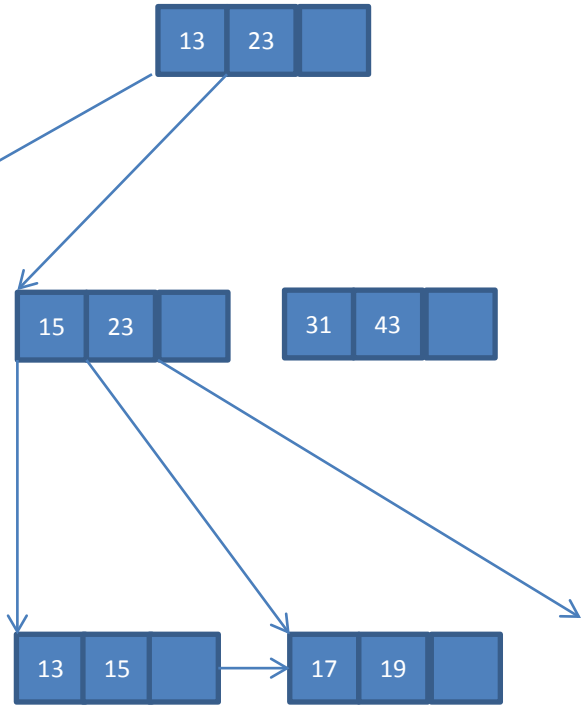
Indexes



- Compare to first key in root, less than, go left
- Compare to first key in 2nd-level 7-node, greater than, compare to 2nd key in 7-node, there is none, go right.
- Compare to first key in 3rd-level 7-node, greater than, compare to 2nd key in 3rd-level 7-node, greater than, compare to 3rd key in 3rd-level 7-node – is none so add

Indexes





Right hand side of original tree should now have this shape

B+-Tree: Maintenance Procedure

- Insertion:
  - Search to determine where item should go
    - A bucket on the last (leaf) level
  - If bucket is not full, add item to it
  - Else
    - Allocate new leaf and move half of discovered bucket's elements to new leaf/bucket
    - If the parent is not full, insert largest component of current leaf into the parent at appropriate place
    - If the parent is full:
      - Split the parent
      - Add the middle key of the current bucket to the parent node
      - Repeat until a parent is found that need not split
  - If root splits, create a new root with one key, two pointer

Fragmentation

- Argue for why fragmentation on PlantLocation is one of the most reasonable ways to fragment the following relation

EMPLOYEE	Name	Title	Salary	PlantLocation
	Joe Steel	Foreman	65000	Edmonton

- What type of fragmentation is the above fragmentation?

Fragmentation

- Argue for why fragmentation on PlantLocation is one of the most reasonable ways to fragment the following relation (assuming a company maintains this database table)

  Likely relation may be split and stored at different sites (those sites would very like match the PlantLocations); name makes no sense to fragment; similar for salary; job maybe is OK to fragment on for managers to deal with – that is fragmentation based on role needs instead of fragmentation for physical reasons

- What type of fragmentation is the above fragmentation?
  HORIZONTAL

Fragmentation

- For the type of fragmentation given, what type of relational operation is employed to generate the fragment?

EMPLOYEE	Name	Title	Salary	PlantLocation
	Joe Steel	Foreman	65000	Edmonton

- What do we need to show to demonstrate the fragmentation is “complete”?

Fragmentation

- For the type of fragmentation given, what type of relational operation is employed to generate the fragment? SELECT (vertical fragmentation uses PROJECT)

- What do we need to show to demonstrate the fragmentation is “complete”?
  - That every row is assigned to a fragment
  - The union of the selection predicates is all of the PlantLocations

Triggers

- Of the times at which triggers can be set to fire, indicate which is most appropriate for implementing the following on a University Student Database:
  - Ensuring an instructor is not assigned to teach two different courses at the same time
  - Ensuring a student cannot register for more than 18 cred hours.
  - Implementing a tuition management system that modifies the students tuition based on the student's current enrollment (add a class, tuition goes up; drop a class, tuition goes down).

Triggers

- Of the times at which triggers can be set to fire, indicate which is most appropriate for implementing the following on a University Student Database:
  - Ensuring an instructor is not assigned to teach two different courses at the same time BEFORE INSERT/UPDATE
  - Ensuring a student cannot register for more than 18 credit hours. BEFORE INSERT/UPDATE
  - Implementing a tuition management system that modifies the students tuition based on the student's current enrollment (add a class, tuition goes up; drop a class, tuition goes down). AFTER INSERT/UPDATE

Query Optimization

- For the three queries below, indicate whether they are equivalent or not. If not, suggest a condition (as general as possible where they are equivalent)

$\sigma_C(\pi_{A_1,A_2,\ldots A_n}(R)) \equiv \pi_{A_1,A_2,\ldots A_n}(\sigma_C(R))$, provided that every attribute involved in C belongs to the set $\{A_1, A_2, \ldots A_n\}$.

$$\sigma_C(R \times S) \equiv \sigma_C(R) \times S.$$

$$\sigma_{C \text{ AND } D}(R) \equiv \sigma_C(\sigma_D(R)).$$

Query Optimization

$\sigma_C(\pi_{A_1,A_2,\ldots A_n}(R)) \equiv \pi_{A_1,A_2,\ldots A_n}(\sigma_C(R))$, provided that every attribute involved in C belongs to the set $\{A_1, A_2, \ldots A_n\}$.

Always true

$$\sigma_C(R \times S) \equiv \sigma_C(R) \times S.$$

False, unless attributes used in selection condition C are constrained to only come from attribute set of R

$$\sigma_{C \text{ AND } D}(R) \equiv \sigma_C(\sigma_D(R)).$$

Always true

Query Optimization

- Given the following relations

```
Branch(BranchName, Assets, City)
Customer(CustomerName, Address, City).
Account(AccountNumber, BranchName, CustomerName, Balance).
```

suggest an equivalent relation that speeds up significantly this expression.

$$\pi_{\text{Asset, BranchName}}(\sigma_{\text{Customer.City='Blacksburg' AND Balance>100000}}(\text{Customer} \bowtie \text{Account} \bowtie \text{Branch}))$$

Query Optimization

Given

```
Branch(BranchName, Assets, City)
Customer(CustomerName, Address, City).
Account(AccountNumber, BranchName, CustomerName, Balance).
```

$$\pi_{\text{Asset, BranchName}}(\sigma_{\text{Customer.City='Blacksburg' AND Balance}>100000}(\text{Customer} \bowtie \text{Account} \bowtie \text{Branch}))$$

a much improved equivalent query is

$$\pi_{\text{Assets,BranchName}}((\pi_{\text{CustomerName}}(\sigma_{\text{City='Blacksburg'}}(\text{Customer})) \bowtie \pi_{\text{BranchName,CustomerName}}(\sigma_{\text{Balance}>100000}(\text{Accounts}))) \bowtie \pi_{\text{Assets,BranchName}}(\text{Branch}))$$

Serializability/Concurrency Control

- Is the following concurrent schedule serializable, and if so, what is the appropriate serial schedule?

read(T1, balx), read(T2, baly), write(T3, balx), read(T2, balx), read(T1, baly)

Yes, {T1,T3,T2}

Serializability/Concurrency Control

- Using 2PL, would this schedule actually occur?

read(T1, balx), read(T2, baly), write(T3, balx),
read(T2, balx), read(T1, baly)

Serializability/Concurrency Control

- Using 2PL, would this schedule actually occur?

read(T1, balx), read(T2, baly), write(T3, balx), read(T2, balx), read(T1, baly)

No… since 2PL only releases locks when all locks are done with, then T1 won't release lock until end of sequence. Thus, T3 won't be able to get exclusive write lock on balx.