Materialized Views

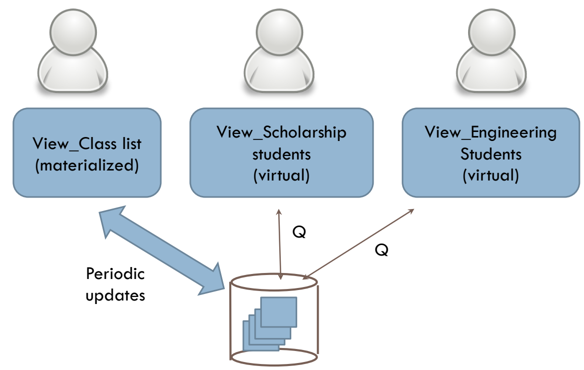
1

- *Materialized* = actually constructed and stored (keeping a temporary table)
- Concerns: maintaining correspondence between the base table and the view when the base table is updated
- Strategy: incremental update

1

Example

2



2

Example: Class Mailing List

3

- The class mailing list db3students is in effect a materialized view of the class enrollment
- Updated periodically
  - You can enroll and miss an email sent out after you enroll.
- Insertion into materialized view normally followed by insertion into base table

3

Materialized View Updates

4

- Update on a single view without aggregate operations: update may map to an update on the underlying base table (most SQL implementations)
- Views involving joins: an update *may map to an* update on the underlying base relations not always possible

4

Example: A Data Warehouse

- Wal-Mart stores every sale at every store in a database.
- Overnight, the sales for the day are used to update a *data warehouse* = materialized views of the sales.
- The warehouse is used by analysts to predict trends and move goods to where they are selling best.

5

INDEXES

6

Example

- Find the price of beers manufactured by Pete's and sold by Joe.

SELECT price

FROM Beers, Sells

WHERE manf = 'Pete''s' AND bar = 'Joe' AND

Sells.beer = Beers.name

7

An Index

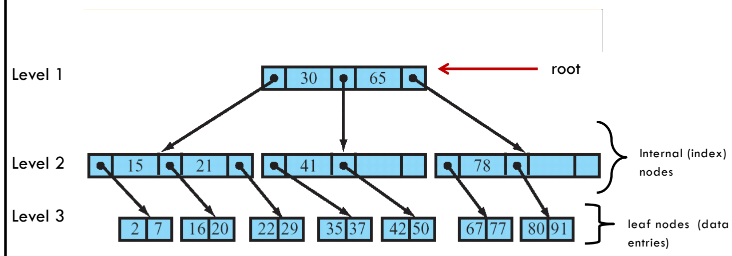
- A data structure used to speed access to tuples of a relation, based on values of one or more attributes ("search key" fields)
- Organizes records via trees or hashing
- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd (on manf) and SellInd (on bar, beer) to find the prices of beers manufactured by Pete's and sold by Joe.

8

B+ Tree Index

9

- The B+ tree structure is the most common index type in databases
- Index files can be quite large, often stored on disk, partially loaded into memory as needed
- Each node is at least 50% full



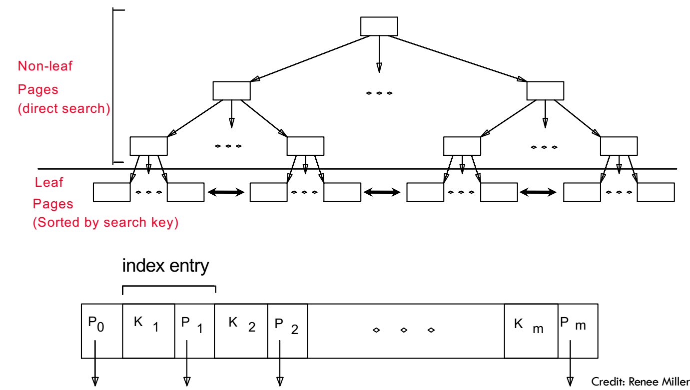
Credit: S. Lee

9

B+ Tree Index

10

Supports equality and range-searches efficiently

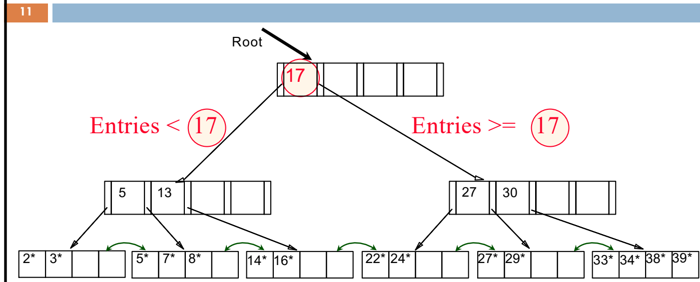


Credit: Renee Miller

10

Example

11

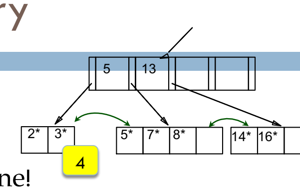


- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

11

Inserting a Data Entry

12



- Find correct leaf L.
- Put data entry onto L.
  - If L has enough space, done!
  - Else, must *split* L (into L and a new node L2)
    - Redistribute entries evenly, copy up middle key.
    - Insert index entry pointing to L2 into parent of L.
- This can happen recursively
  - To split index node, redistribute entries evenly, but push up middle key.
- Splits “grow” tree; root split increases height.

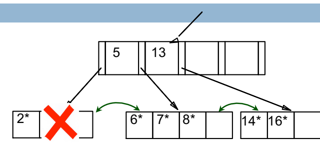
Insert data value 4

12

Deleting a Data Entry

Delete value 3

13



- Start at root, find leaf L where entry belongs.
- Remove the entry.
  - If L is at least half-full, done!
  - If not,
    - Try to re-distribute, borrowing from sibling (adjacent node with same parent as L).
    - If re-distribution fails, merge L and sibling.
- If merge occurred, must delete entry (pointing to L or sibling) from parent of L.
- Merge could propagate to root, decreasing height.

13

Balanced vs. Unbalanced Trees

14

- In a balanced tree, every path from the root to a leaf node is the same length.

- Balanced

```
          [===]
          /    \
     [===]      [===]
     /   \      /   \
[===] [===] [===] [===]
```

- Unbalanced

```
                  [===]
                  /
             [===]
             /   \
        [===]   [===]
        /
   [===]
   /    \
[===] [===]
```

Credit: S. Lee

14

Hash Based Indexes

15

- Good for equality selections.
- Index is a collection of buckets
  - Bucket = primary page plus zero or more overflow pages.
  - Buckets contain data entries.
- Hashing function h: h(r) = bucket in which (data entry for) record r belongs. h looks at the search key fields of r.
  - No need for "index entries" in this scheme.

15

Index Classification

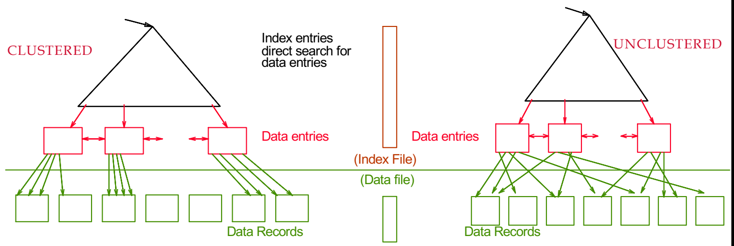
16

- Primary vs. secondary: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- Clustered vs. unclustered: If order of index data entries is the same as order of data records, then called clustered index.
  - A table can have at most one clustered index – why?

16

Clustered vs. Unclustered Index

17



17

Declaring Indexes

18

- No standard!
- Typical syntax:

```
CREATE INDEX BeerInd ON Beers(manf);
CREATE INDEX SellInd ON Sells(bar,
  beer);
```

18

Using Indexes

19

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd and SellInd to find the prices of beers manufactured by Pete's and sold by Joe.

19

Using Indexes --- (2)

20

```
SELECT price
FROM Beers, Sells
WHERE manf = 'Pete''s' AND
   Beers.name = Sells.beer AND
   bar = 'Joe''s Bar';
```

1. Use BeerInd to get all the beers made by Pete's.
2. Then use SellInd to get prices of those beers, with bar = 'Joe''s Bar'

20

Understanding the Workload

21

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

21

Choice of Indexes

22

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

22

Choice of Indexes (cont'd)

23

- One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index.
  - Implies an understanding of how a DBMS evaluates queries and creates query evaluation plans.
- Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

23

Guidelines

24

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
- Try to choose indexes that benefit as many queries as possible.
  - Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

24

Examples

25

- B+ tree index on E.age can be used to get qualifying tuples.

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

- Equality queries and duplicates:
  - Indexing on E.hobby

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby='Stamps'
```

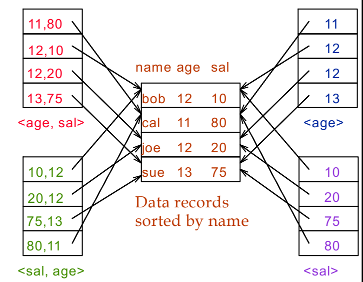
25

Composite Search Keys

26

- Composite Search Keys: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal =75
  - Range query:
    - age=20 and sal > 10
- Data entries in index sorted by search key to support range queries.

Examples of composite key indexes



Data entries in index sorted by <sal,age>

Data entries sorted by <sal>

26