# Index tuning

Vu Tuyet Trinh
trinhvt@soict.hust.edu.vn

Department of Information Systems
SoICT-HUST

# *Copyrights:*

*Many slides belong to the tutorial:*
Database Tuning
*Principles, Experiments and Troubleshooting Techniques*
Dennis Shasha (shasha@cs.nyu.edu)
Philippe Bonnet (bonnet@diku.dk)

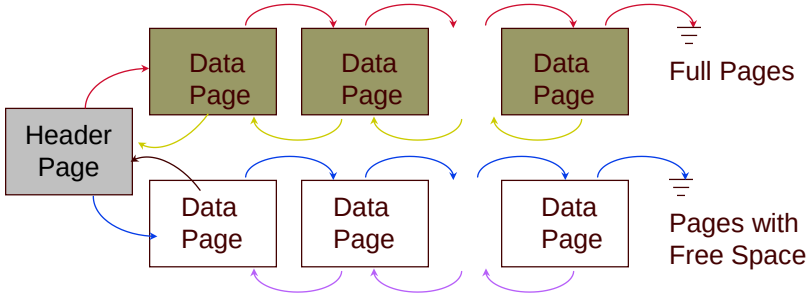
and some from the web …

# File Organization

- Data storage in file
  - records, blocks and access structures
- Organisation
  - Heap files: for full file scans or frequent updates
    - Data unordered
    - Write new data at end
  - Index: if retrieved in key attributes
    - Need to store index
  - Sorted Files: if retrieved in sort order or want range
    - Need external sort or an index to keep sorted
  - Hashed Files: if selection on equality
    - Collection of buckets with primary & overflow pages
    - Hashing function over search key attributes

# Heap File

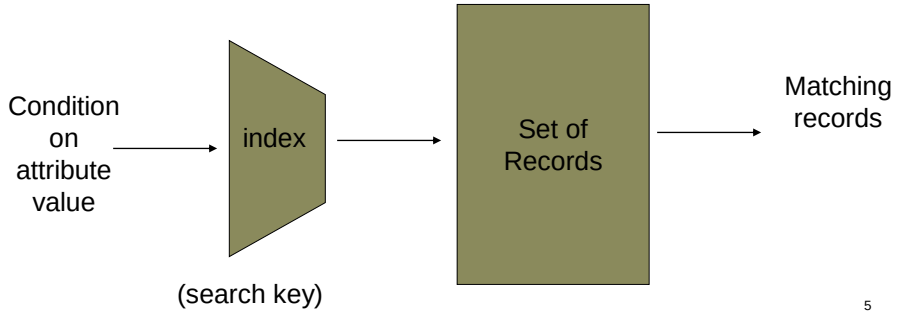
- Organization
  - Data unordered
  - Write new data at end




Need a full scan file for Search, Insert, Update, Delete operations

# Index

An index is a data structure that supports efficient access to data



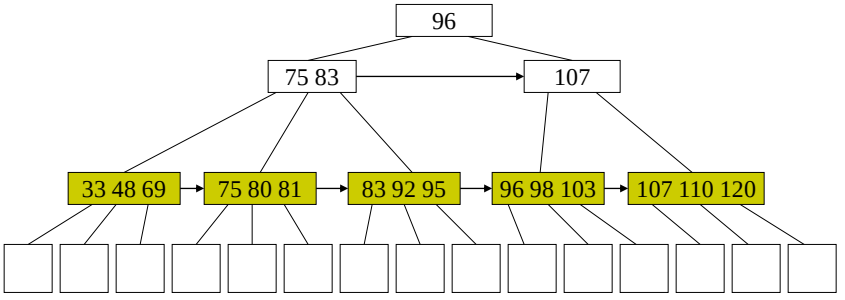

5

# Classes of Indexes

- Primary vs. secondary: primary has primary key
- Clustered vs. unclustered: order of records and index approximately same
  - Alternative 1 implies clustered, but not vice-versa
  - A file can be clustered on at most one search key
- Dense vs. Sparse: dense has index entry per data value; sparse may "skip" some
  - Alternative 1 always leads to dense index
  - Every sparse index is clustered!
  - Sparse indexes are smaller; however, some useful optimizations are based on dense indexes

# B+-Tree

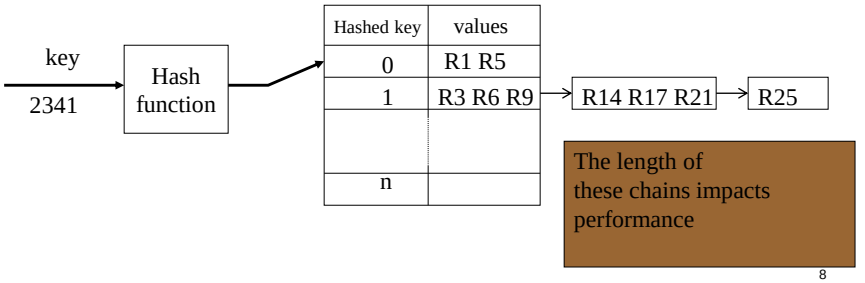
- A B+-Tree is a balanced tree whose leaves contain a sequence of key-pointer pairs.




7

# Hash Index

- A hash index stores key-value pairs based on a pseudo-randomizing function called a *hash function*.

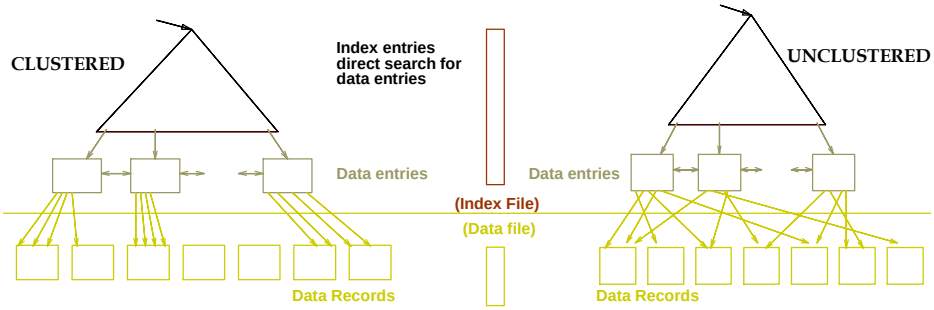



8

# Clustered vs. Unclustered Index

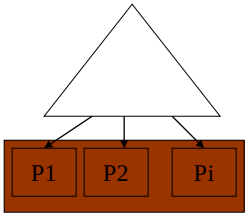
Suppose Index Alternative (2) used, records are stored in Heap file

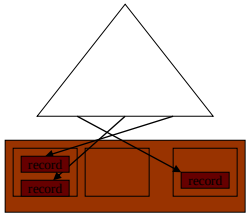
- Perhaps initially sort data file, leave some gaps
- Inserts may require overflow pages




# Dense / Sparse Index

- Sparse index
  - Pointers are associated to pages
- Dense index
  - Pointers are associated to records
  - Non clustered indexes are dense




10

# Index Implementations in some major DBMS

- SQL Server
  - B+-Tree data structure
  - Clustered indexes are sparse
  - Indexes maintained as updates/insertions/delete s are performed
- DB2
  - B+-Tree data structure, spatial extender for R-tree
  - Clustered indexes are dense
  - Explicit command for index reorganization
- Oracle
  - B+-tree, hash, bitmap, spatial extender for R-Tree
  - No clustered index until 10g
    - Index organized table (unique/clustered)
    - Clusters used when creating tables.
- MySQL
  - B+-Tree, R-Tree (geometry and pairs of integers)
  - Indexes maintained as updates/insertions/delete s are performed



# Constraints and Indexes

- Primary Key, Unique
  - A non-clustered index is constructed on the attribute(s) that compose the primary key with the constraint that values are unique.
- Foreign Key
  - By default, no index is created to enforce a foreign key constraint.

# Performance Issues

- Type of Query
- Index Overhead



# Types of Queries

1. Point Query

```
SELECT balance
FROM accounts
WHERE number = 1023;
```

2. Multipoint Query

```
SELECT balance
FROM accounts
WHERE branchnum = 100;
```

3. Range Query

```
SELECT number
FROM accounts
WHERE balance > 10000;
```

4. Prefix Match Query

```
SELECT *
FROM employees
WHERE  name = 'Jensen'
       and firstname = 'Carl'
       and age < 30;
```

# Types of Queries

5. Extremal Query

```
SELECT *
FROM accounts
WHERE balance =
  max(select balance from
accounts)
```

6. Ordering Query

```
SELECT *
FROM accounts
ORDER BY balance;
```

7. Grouping Query

```
SELECT branchnum,
avg(balance)
FROM accounts
GROUP BY branchnum;
```

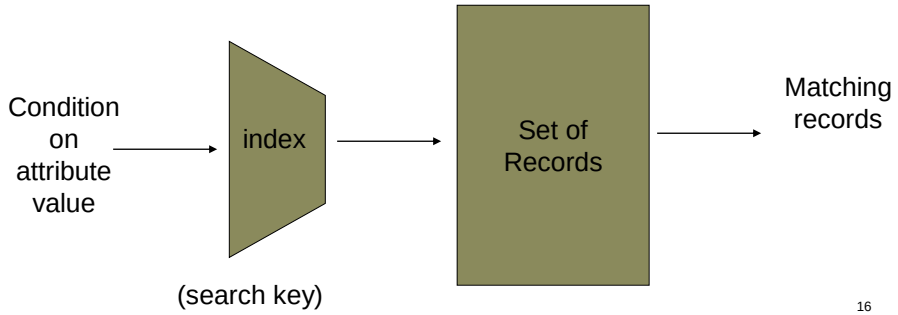
8. Join Query

```
SELECT distinct branch.adresse
FROM accounts, branch
WHERE
  accounts.branchnum =
                branch.number
and accounts.balance > 10000;
```

15

# Index

An index is a data structure that supports efficient access to data




16

## Index Tuning Knobs

- Index data structure
- Search key
- Size of key
- Clustered/Non-clustered/No index
- Covering

17

## Clustered Index

Benefits of a clustered index:

1. A sparse clustered index stores fewer pointers than a dense index.
   - This might save up to one level in the B-tree index.
2. A clustered index is good for multipoint queries
   - White pages in a paper telephone book
3. A clustered index based on a B-Tree supports range, prefix, extremal and ordering queries well.

18

# Clustered Index

4. A clustered index (on attribute X) can reduce lock contention:

   Retrieval of records or update operations using an equality, a prefix match or a range condition based on X will access and lock only a few consecutive pages of data

Cost of a clustered index

1. Cost of overflow pages
   - Due to insertions
   - Due to updates (e.g., a NULL value by a long string)

19

# Clustered Index

- Because there is only one clustered index per table, it might be a good idea to replicate a table in order to use a clustered index on two different attributes
  - Yellow and white pages in a paper telephone book
  - Low insertion/update rate

20

# Non-Clustered Index

Benefits of non-clustered indexes

1. A dense index can eliminate the need to access the underlying table through covering.
   - It might be worth creating several indexes to increase the likelihood that the optimizer can find a covering index
2. A non-clustered index is good if each query retrieves significantly fewer records than there are pages in the table.
   - Point queries
   - Multipoint queries:

*number of distinct key values > c * number of records per page*

*Where c is the number of pages retrieved in each prefetch*



# Index on Small Tables

- Tuning manuals suggest to avoid indexes on small tables
  - If all data from a relation fits in one page then an index page adds an I/O
  - If each record fits in a page then an index helps performance

# Key Compression

- Use key compression
  - If you are using a B-tree
  - Compressing the key will reduce the number of levels in the tree
  - The system is not CPU-bound
  - Updates are relatively rare



# Summary

1. Use a hash index for point queries
   Use a B-tree for multipoint and range queries
2. Use clustering
   - if your queries need all or most of the fields of each records returned
   - if range queries are asked
3. Use a dense index to cover critical queries
4. Don't use an index if the time lost when inserting and updating overwhelms the time saved when querying