Data storage

Tuesday, May 7, 2019 6:15 PM

- Hard Disk mechanical devices
  - block. → 4 ~ 16 GB unit of read/write ⇒ indispensible; always read one block.
  - Once in memory, we call it a page.
  - { rotation speed: not increase.
    density: slowly increased
  - CPU once read 1/4/8 byte.
  - Read at rotation speed
    - fast: Sequential read
    - slow: random read
  - Rule of thumb
    - random read 1% ~ 2% of file = sequential read the whole file.
    - 1~2% decrease over time because of increased memory.
- Data Storage
  - DBMS store data in files
  - On disk, a file is split into blocks

- Heap file: unsorted
- Sequential file: sorted according to key

- **Index**: neither Heap file nor sequential file.
  - def: an additional file that allows fast access to records in the data file given a search key.
  - Contains: (key, value) pairs
    - key = an attribute vale → here means search key
    - value: a pointer to the record or the record itself.

- key:
  - Primary key: unique
  - Key of a sequential file: how file is sorted.
  - Index key: how index is organized

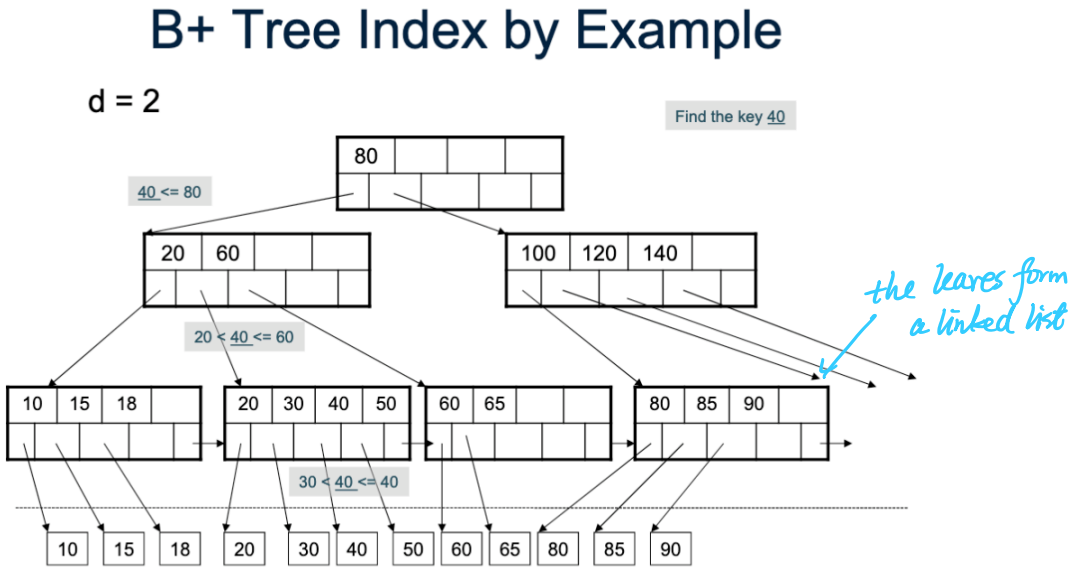
  - Index can be:
    - Dense = one entry per record
    - Sparse = one entry per block

Binary search on index

e.g.2. Index on fName

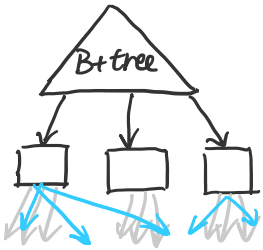
Index can only be dense. (∵ in data file, data is not put in block by fName)

- index organization
  - Hash table
  - B+ trees — all the keys are on the leaves (difference between Btree)
    - Every node can have more than one branch — max number of branch is designed on node
    - search tree but not binary instead have higher fan-out
    - leaves form a linked list



  - Specialized indices: bitmaps, R-trees, inverted index

- Clustered / Unclustered



— clustered
— unclustered

- the key of data file must be the same as index.
- We can only have 1 clustered index. → file can only be sorted once.

  and a lot of unclustered indices.
- Clustered: records close in index are close in data

  Unclustered: records closed in index may be far in data
- Primary / Secondary

  Meaning I: P → primary key. S: otherwise.

  Meaning II: P → clustered Secondary: unclustered.

- Selection
  - "on the fly"
  - Index-based selection

logical plan:

σ_{300}

|

Takes

```
SELECT *
FROM Takes y
WHERE y.courseID=300;
```

Physical Plan 1:

onthefly σ_{300}

|

Takes

```
for y in Takes
  if courseID=300 then
    output y
```

Physical Plan 2:

index-based σ_{300}

|

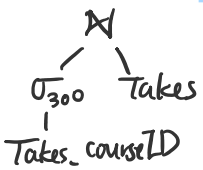
Takes

```
Take_courseID=
    index on Takes.courseID)
rid_list = Take_courseID).get(300)
for r in rid_list
    y=student.getRecord(rid)
```

SQL Server ↓

represents index-based select as a join

output y



- Join Nested-Loop Join vs Index-based Join

```
CREATE TABLE V(M int, N text, P int);
CREATE INDEX V1 ON V(N);
CREATE INDEX V2 on V(P,M);
```

order matters

```
CREATE UNIQUE INDEX V4 ON V(N)
```

← the attributes are unique

CLUSTERED) (Not support by SQLite)

↑ generate a clustered file

- If there're no updates, just create an index for all attributes.

For range queries, use BTREE. e.g. N>15 N<15000.

e.g.
```
select *
from V
where N=?
```
100000 queries

```
select *
from V
where P=?
```
100 queries

↓ if there're 10000 queries,

try using V(N) primary index and V(P) secondary index

```
insert into V
values (???)
```
200 queries

if there are updates, only create index on N.

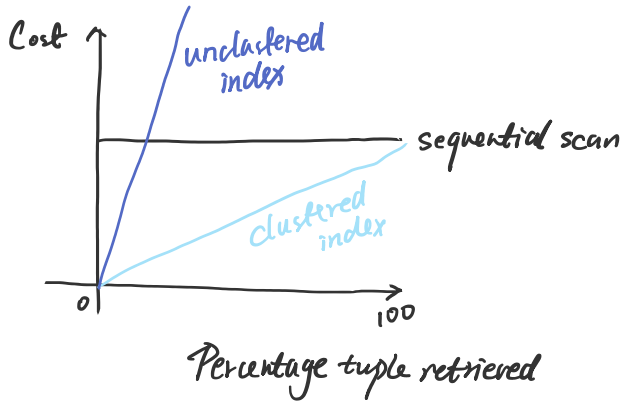
if not, create index on both N and P.

- Two typical kinds of queries
  - Point queries P=?
  - Range queries P>? and P<?

- Hash or B^+ tree index
- Clustered or not

- B^+ tree index
- Clustered

- Range queries benefit mostly from clustering because they may read more than 1-2%.



select *
from R
where R.K>? and R.k<?

- Summary for physical plan:

  Access path selection — scan the relation vs index

  Implementation choice — (for operator)

  Scheduling decisions — pipelined vs intermediate materialization

  things go materialization always go blocked.
  ⇒ they are materialized into an intermediate table.
  e.g. group by.