CS 525: Advanced Database Organization

05: Hashing and More

Boris Glavic

Slides: adapted from a course taught by Hector Garcia-Molina, Stanford InfoLab

Hashing

- key → h(key)

Buckets (typically 1 disk block)

Two alternatives

(1) key → h(key)

records

(2) key → h(key)

Index

Example hash function

- Key = ‘x1 x2 ... xn’ n byte character string
- Have b buckets
- h: add x1 + x2 + ..... xn
  - compute sum modulo b

• Alt (2) for “secondary” search key
This may not be best function ...
Read Knuth Vol. 3 if you really need to select a good function.

Within a bucket:
• Do we keep keys sorted?
• Yes, if CPU time critical & Inserts/Deletes not too frequent

Good hash function:
Expected number of keys/bucket is the same for all buckets

Next: example to illustrate inserts, overflows, deletes

EXAMPLE 2 records/bucket

case
\begin{array}{|c|c|}
\hline
0 & 1 \\
1 & 2 \\
2 & 3 \\
\hline
\end{array}

h(a) = 1
h(b) = 2
h(c) = 1
h(d) = 0

EXAMPLE 2 records/bucket

case
\begin{array}{|c|c|}
\hline
0 & d \\
1 & a \\
2 & c \\
3 & b \\
\hline
\end{array}

h(a) = 1
h(b) = 2
h(c) = 1
h(d) = 0
h(e) = 1
EXAMPLE: 2 records/bucket

INSERT:
- h(a) = 1
- h(b) = 2
- h(c) = 1
- h(d) = 0
- h(e) = 1

EXAMPLE: deletion

Delete:
- e
- f
- c
- g

Rule of thumb:
- Try to keep space utilization between 50% and 80%
  - Utilization = \( \frac{\text{# keys used}}{\text{total # keys that fit}} \)

Rule of thumb:
- Try to keep space utilization between 50% and 80%
  - Utilization = \( \frac{\text{# keys used}}{\text{total # keys that fit}} \)
- If < 50%, wasting space
- If > 80%, overflows significant
  - depends on how good hash function is & on # keys/bucket
How do we cope with growth?

- Overflows and reorganizations
- Dynamic hashing

Extensible hashing: two ideas

(a) Use $i$ of $b$ bits output by hash function

$$h(K) \rightarrow \underbrace{00110101}_{b}$$

use $i \rightarrow$ grows over time...

(b) Use directory $h(K)[i]$ to bucket

Example: $h(k)$ is 4 bits; 2 keys/bucket

- $i = 1$:
  - $0001$
  - $1001$
  - $1100$

  Insert 1010

Example: $h(k)$ is 4 bits; 2 keys/bucket

- $i = 1$:
  - $0001$
  - $1001$
  - $1100$

  Insert 1010
Example: \( h(k) \) is 4 bits; 2 keys/bucket

Insert 1010

New directory

Example continued

Insert:

Input continued

Insert:

Input continued

Insert:

Input continued

Insert:

Input continued
Extensible hashing: deletion
- No merging of blocks
- Merge blocks and cut directory if possible (Reverse insert procedure)

Deletion example:
- Run thru insert example in reverse!

Note: Still need overflow chains
- Example: many records with duplicate keys

Solution: overflow chains
- Insert 1100
- Add overflow block:

Summary
Extensible hashing
- Can handle growing files
- With less wasted space
- With no full reorganizations
Summary
Extensible hashing
+ Can handle growing files
  - with less wasted space
  - with no full reorganizations
+ Indirection
  (Not bad if directory in memory)
+ Directory doubles in size
  (Now it fits, now it does not)

Linear hashing
- Another dynamic hashing scheme

Two ideas:
(a) Use $i$ low order bits of hash

(b) File grows linearly

Rule
If $h(k)[i] \leq m$, then
look at bucket $h(k)[i]$.
else, look at bucket $h(k)[i] - 2^{i-1}$

Example
$b=4$ bits, $i=2$, 2 keys/bucket

Future growth buckets

if $h(k)[i] \leq m$, then
look at bucket $h(k)[i]$.
else, look at bucket $h(k)[i] - 2^{i-1}$

Example
$b=4$ bits, $i=2$, 2 keys/bucket

Future growth buckets

if $h(k)[i] \leq m$, then
look at bucket $h(k)[i]$.
else, look at bucket $h(k)[i] - 2^{i-1}$

* insert 0101
**Example**  
$b=4$ bits,  $i=2$, 2 keys/bucket  

- Insert 0101  
- Can have overflow chains!

- **Rule**  
  If $h(k)[i] = m$, then look at bucket $h(k)[i]$  
  else, look at bucket $h(k)[i] - 2^{i-1}$

- **Note**  
  - In textbook, $n$ is used instead of $m$  
  - $n=m+1$

**Future growth buckets**

---

**Example**  
$b=4$ bits,  $i=2$, 2 keys/bucket  

**Example**  
$b=4$ bits,  $i=2$, 2 keys/bucket
Example: $b=4$ bits, $i=2$, 2 keys/bucket

- Insert 0101

Future growth buckets

$m = 01$ (max used block)

Example Continued: How to grow beyond this?

$i = 2$

Future growth buckets

$m = 11$ (max used block)

Example Continued: How to grow beyond this?

$i = 2^3$

Future growth buckets

$m = 11$ (max used block)

Example Continued: How to grow beyond this?

$i = 2^3$

Future growth buckets

$m = 11$ (max used block)

Example Continued: How to grow beyond this?

$i = 2^3$

Future growth buckets

$m = 11$ (max used block)

When do we expand file?

- Keep track of: $\frac{\text{# used slots}}{\text{total # of slots}} = U$
When do we expand file?

- Keep track of: \( \frac{\text{# used slots}}{\text{total # of slots}} = U \)
- If \( U > \text{threshold} \) then increase \( m \) (and maybe \( i \))

Example: BAD CASE

<table>
<thead>
<tr>
<th>Very full</th>
<th>Very empty</th>
<th>Need to move</th>
<th>Would waste space...</th>
</tr>
</thead>
</table>

Summary

- Linear Hashing

  - Can handle growing files
    - with less wasted space
    - with no full reorganizations
  - No indirection like extensible hashing
  - Can still have overflow chains

Hashing

- How it works
  - Dynamic hashing
    - Extensible
    - Linear

Indexing vs Hashing

- Hashing good for probes given key
  - e.g., `SELECT ... FROM R WHERE R.A = 5`
  - Point Queries

Next:

- Indexing vs Hashing
- Index definition in SQL
- Multiple key access
Indexing vs Hashing

• INDEXING (Including B Trees) good for Range Searches:
  e.g., SELECT FROM R WHERE R.A > 5

-> Range Queries

Index definition in SQL

• Create index name on rel (attr)
• Create unique index name on rel (attr)
  defines candidate key
• Drop INDEX name

Note
CANNOT SPECIFY TYPE OF INDEX
(e.g. B-tree, Hashing, ...)
OR PARAMETEES
(e.g. Load Factor, Size of Hash,...)

... at least in standard SQL...

Vendor specific extensions allow that

Multi-key Index

Motivation: Find records where
DEPT = “Toy” AND SAL > 50k

Strategy I:
• Use one index, say Dept.
• Get all Dept = “Toy” records
  and check their salary
Strategy II:
- Use 2 Indexes; Manipulate Pointers

Toy: ~

Strategy III:
- Multiple Key Index

One idea:

Example:

Record:
- Name=Joe
- DEPT=Sales
- SAL=15k

For which queries is this index good?
- Find RECs Dept = "Sales" \( \land \) SAL=20k
- Find RECs Dept = "Sales" \( \land \) SAL > 20k
- Find RECs Dept = "Sales"
- Find RECs SAL = 20k

Interesting application:
- Geographic Data

Queries:
- What city is at \( <X_i,Y_i> \)?
- What is within 5 miles from \( <X_i,Y_i> \)?
- Which is closest point to \( <X_i,Y_i> \)?
Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

• Search points near f
• Search points near b
Queries

- Find points with \( Y_i > 20 \)
- Find points with \( X_i < 5 \)
- Find points “close” to \( i = <12,38> \)
- Find points “close” to \( b = <7,24> \)

Next

- Even more index structures 😊