CS 277: Database System Implementation

Notes 5: Hashing and More

Arthur Keller
Hashing

current → h(key)

[key]

Buckets (typically 1 disk block)
Two alternatives

(1) $\text{key} \rightarrow h(\text{key})$

\begin{center}
\begin{tabular}{|c|}
\hline
records \\
\hline
\end{tabular}
\end{center}
Two alternatives

(2) key $\rightarrow h(\text{key})$

• Alt (2) for "secondary" search key
Example hash function

- Key = ‘x₁ x₂ ... xₙ’  n byte character string
- Have b buckets
- h: add x₁ + x₂ + .... + xₙ
  - compute sum modulo b
“This may not be best function ...
“Read Knuth Vol. 3 if you really need to select a good function.

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

• Do we keep keys sorted?

• Yes, if CPU time critical
  & Inserts/Deletes not too frequent
Next: example to illustrate inserts, overflows, deletes

\[ h(K) \]
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
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
  - Linear
Extensible hashing: two ideas

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

$h(K) \rightarrow 00110101$

use $i \rightarrow$ grows over time....
(b) Use directory

\[ h(K)[i] \] to bucket
Example: \( h(k) \) is 4 bits; 2 keys/bucket

Insert 1010

New directory
Example continued

Insert:

0111
0000
Example continued

Insert:

1001
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!
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 the low order bits of hash

(b) File grows linearly
**Example**  \( b=4 \) bits,  \( i=2 \),  \( 2 \) keys/bucket

\[
\begin{array}{c}
0000 \\
1010
\end{array}
\quad
\begin{array}{c}
0101 \\
1111
\end{array}
\quad
\begin{array}{c}
00 \\
01
\end{array}
\quad
\begin{array}{c}
10 \\
11
\end{array}
\]

\( m = 01 \) (max used block)

- insert 0101
- can have overflow chains!

**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

\[\begin{array}{c|c|c}
00 & 0000 & 0101 \\
1010 & 0101 & 1111 \\
\end{array}\]

\[\begin{array}{c|c|c}
01 & 1010 & 1010 \\
10 & 0101 & 1111 \\
11 & 0101 & 1111 \\
\end{array}\]

\( m = 01 \) (max used block)

Future growth buckets

• insert 0101
Example Continued: How to grow beyond this?

\[ i = 2 \]

\[ m = 11 \text{ (max used block)} \]
* When do we expand file?

- Keep track of: \[
\frac{\text{\# used slots}}{\text{total \# of slots}} = U
\]

- If \( U > \) threshold then increase \( m \) (and maybe \( i \) )
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
Example: BAD CASE

Very full

Very empty

Need to move $m$ here...
Would waste space...
Summary

Hashing
  - How it works
  - Dynamic hashing
    - Extensible
    - Linear
Next:

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

• Hashing good for probes given key
e.g., SELECT ...
   FROM R
   WHERE R.A = 5
Indexing vs Hashing

- INDEXING (Including B Trees) good for Range Searches:
  
e.g.,
  
  ```sql
  SELECT
  FROM R
  WHERE R.A > 5
  ```
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 PARAMETERS (e.g. Load Factor, Size of Hash, ...) ... at least in SQL...
Note \textbf{ATTRIBUTE LIST} \Rightarrow \textbf{MULTIKEY INDEX}

(next)
e.g., \textbf{CREATE INDEX} foo \textbf{ON} R(A,B,C)
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 → □□□□□ □□□□□ □□□□□ ← Sal
> 50k
Strategy III:

- Multiple Key Index

One idea:
Example

Dept Index

Art
Sales
Toy

Salary Index

10k
15k
17k
21k

12k
15k
15k
19k

Example Record

Name=Joe
DEPT=Sales
SAL=15k
For which queries is this index good?

- Find RECs Dept = “Sales” ∧ SAL=20k
- Find RECs Dept = “Sales” ∧ SAL ≥ 20k
- Find RECs Dept = “Sales”
- Find RECs SAL = 20k
Interesting application:

- Geographic Data

DATA:

$$<X_1, Y_1, \text{Attributes}>$$

$$<X_2, Y_2, \text{Attributes}>$$

...
Queries:

• What city is at \(<Xi,Yi>\)?
• What is within 5 miles from \(<Xi,Yi>\)?
• Which is closest point to \(<Xi,Yi>\)?
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 $b = <7,24>$
• Many types of geographic index structures have been suggested
  • Quad Trees
  • R Trees
Two more types of multi key indexes

- Grid
- Partitioned hash
Grid Index

Key 1

V_1
V_2
V_n

Key 2

X_1  X_2  ......  X_n

To records with key1=V_3, key2=X_2
CLAIM

• Can quickly find records with
  – key 1 = $V_i$ $\land$ Key 2 = $X_j$
  – key 1 = $V_i$
  – key 2 = $X_j$

• And also ranges....
  – E.g., key 1 $\geq$ $V_i$ $\land$ key 2 < $X_j$
* But there is a **catch** with Grid Indexes!

- How is Grid Index stored on disk?

  Like

  Array...

  V1
  
  X1 X2 X3 X4
  
  V2
  
  X1 X2 X3 X4
  
  V3
  
  X1 X2 X3 X4

**Problem:**

- Need regularity so we can compute position of $<Vi,Xj>$ entry
Solution: Use Indirection

*Grid only contains pointers to buckets*
With indirection:

• Grid can be regular without wasting space
• We do have price of indirection
Can also index grid on value ranges

**Salary**

<table>
<thead>
<tr>
<th>Salary Range</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-20K</td>
<td>1</td>
</tr>
<tr>
<td>20K-50K</td>
<td>2</td>
</tr>
<tr>
<td>50K-∞</td>
<td>3</td>
</tr>
</tbody>
</table>

**Grid**

<table>
<thead>
<tr>
<th>Linear Scale</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Toy</th>
<th>Sales</th>
<th>Personnel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Grid files

- Good for multiple-key search
- Space, management overhead
  (nothing is free)
- Need partitioning ranges that evenly split keys
Partitioned hash function

Idea: 010110 1110010

Key1 → h1  h2 ← Key2
EX:

h1(toy) = 0
h1(sales) = 1
h1(art) = 1

h2(10k) = 01
h2(20k) = 11
h2(30k) = 01
h2(40k) = 00

<Fred,toy,10k>, <Joe,sales,10k>, <Sally,art,30k>

Insert
<table>
<thead>
<tr>
<th>h1(toy)</th>
<th>h1(sales)</th>
<th>h1(art)</th>
<th>h2(10k)</th>
<th>h2(20k)</th>
<th>h2(30k)</th>
<th>h2(40k)</th>
<th>Find Emp. with Dept. = Sales ∧ Sal=40k</th>
</tr>
</thead>
<tbody>
<tr>
<td>=0</td>
<td>=1</td>
<td>=1</td>
<td>=01</td>
<td>=11</td>
<td>=01</td>
<td>=00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( h_1(\text{toy}) = 0 \)
\( h_1(\text{sales}) = 1 \)
\( h_1(\text{art}) = 1 \)

\( h_2(10k) = 01 \)
\( h_2(20k) = 11 \)
\( h_2(30k) = 01 \)
\( h_2(40k) = 00 \)

- Find Emp. with Sal=30k

<table>
<thead>
<tr>
<th></th>
<th>&lt;Fred&gt;</th>
<th>&lt;Joe&gt;</th>
<th>&lt;Jan&gt;</th>
<th>&lt;Mary&gt;</th>
<th>&lt;Sally&gt;</th>
<th>&lt;Tom&gt;</th>
<th>&lt;Bill&gt;</th>
<th>&lt;Andy&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h1(toy)</td>
<td>0</td>
<td>000</td>
<td>&lt;Fred&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>-----</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h1(sales)</td>
<td>1</td>
<td>001</td>
<td>&lt;Joe&gt;&lt;Jan&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h1(art)</td>
<td>1</td>
<td>010</td>
<td>&lt;Mary&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>h2(10k)</th>
<th>01</th>
<th>100</th>
<th>&lt;Sally&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2(20k)</td>
<td>11</td>
<td>101</td>
<td>&lt;Tom&gt;&lt;Bill&gt;</td>
</tr>
<tr>
<td>h2(30k)</td>
<td>01</td>
<td>110</td>
<td>&lt;Andy&gt;</td>
</tr>
<tr>
<td>h2(40k)</td>
<td>00</td>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>

- Find Emp. with Dept. = Sales
Summary

Post hashing discussion:

- Indexing vs. Hashing
- SQL Index Definition
- Multiple Key Access
  - Multi Key Index
    - Variations: Grid, Geo Data
- Partitioned Hash
Reading Chapter 14

• Skim the following sections:
  – 14.3.6, 14.3.7, 14.3.8
  – 14.4.2, 14.4.3, 14.4.4

• Read the rest
The BIG picture....

- Chapters 11 & 12: Storage, records, blocks...
- Chapter 13 & 14: Access Mechanisms
  - Indexes
  - B trees
  - Hashing
  - Multi key
- Chapter 15 & 16: Query Processing