# CS 245: Database System Principles

## Notes 5: Hashing and More

Hector Garcia-Molina

---

### Hashing

\[ \text{key} \rightarrow h(\text{key}) \]

- **Buckets** (typically 1 disk block)

---

### Two alternatives

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

   - \( \cdots \)
   - records
   - \( \cdots \)

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

   - Index
   - record

---

### Example hash function

- Key = ‘\(x_1\ x_2\ \ldots\ x_n\)’ \(n\) byte character string
- Have \(b\) buckets
- \(h:\) add \(x_1 + x_2 + \ldots + x_n\)
  - 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.

Good hash function: 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

EXAMPLE 2 records/bucket

<table>
<thead>
<tr>
<th>INSERT:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(a) = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(b) = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(c) = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(d) = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLE 2 records/bucket

<table>
<thead>
<tr>
<th>INSERT:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(a) = 1</td>
<td></td>
<td>a</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>h(b) = 2</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(c) = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(d) = 0</td>
<td></td>
<td>e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h(e) = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE 2 records/bucket

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

EXAMPLE: deletion

**DELETE:**
- \( e \)
- \( f \)
- \( g \)

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

\[ h(K) \rightarrow \text{00110101} \]
use $i \rightarrow$ grows over time...

(b) Use directory

\[ h(K)[i] \rightarrow \text{to bucket} \]

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

\[ i = \begin{cases} 1 \\ 0001 \\ 1001 \\ 1100 \end{cases} \]

Insert 1010

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

\[ i = \begin{cases} 1 \\ 0001 \\ 1001 \\ 1100 \end{cases} \]

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

Insert 1010

Example continued

Example continued

Example continued

Example continued

Example continued
Example continued

<table>
<thead>
<tr>
<th>i = 2</th>
<th>000</th>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1001</td>
<td>1010</td>
</tr>
</tbody>
</table>

Insert: 1001

```
i = 3
000
001
010
011
100
101
110
111
```

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
  insert 1100
  if we split:

```
1
1101
1100
```

Solution: overflow chains
- insert 1100
- add overflow block:

```
1
1101
1100
```

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

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

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

- Rule

- Insert 0101
Example \( b=4 \) bits, \( i=2 \), 2 keys/bucket

- insert 0101
- can have overflow chains!

\[
\begin{array}{cccc}
00 & 01 & 10 & 11 \\
0000 & 0101 & 1011 & \\
1010 & 0000 & 1000 & \\
\end{array}
\]

Future growth buckets

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

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

Note

- In textbook, \( n \) is used instead of \( m \)
- \( n=m+1 \)

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

Future growth buckets

\( n=10 \)

• can have overflow chains!
Example: \( b = 4 \) bits, \( i = 2 \), 2 keys/bucket

\[
\begin{array}{cccc}
0000 & 0101 & 1010 & 1111 \\
10 & 10 & 11 & \\
\end{array}
\]

Future growth buckets

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

Example Continued: How to grow beyond this?

\( i = 2 \)

\[
\begin{array}{cccc}
0000 & 0101 & 1010 & 1111 \\
00 & 01 & 10 & 11 \\
\end{array}
\]...

\( m = 11 \) (max used block)

Example Continued: How to grow beyond this?

\( i = 2^3 \)

\[
\begin{array}{cccc}
0000 & 0101 & 1010 & 1111 \\
00 & 01 & 10 & 11 \\
100 & 101 & 110 & 111 \\
\end{array}
\]...

\( m = 11 \) (max used block)

Example Continued: How to grow beyond this?

\( i = 2^3 \)

\[
\begin{array}{cccc}
0000 & 0101 & 1010 & 1111 \\
00 & 01 & 10 & 11 \\
100 & 101 & 110 & 111 \\
100 & & & \\
\end{array}
\]...

\( m = 2^3 \) (max used block)

Example Continued: How to grow beyond this?

\( i = 2^3 \)

\[
\begin{array}{cccc}
0000 & 0101 & 1010 & 1111 \\
00 & 01 & 10 & 11 \\
100 & 101 & 110 & 111 \\
100 & & & \\
\end{array}
\]...

\( m = 2^3 \) (max used block)

When do we expand file?

- Keep track of: \( \# \) used slots, 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 \( i \))

**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., 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

ATTRIBUTE LIST ⇒ MULTIKEY INDEX
(next)
e.g., CREATE INDEX foo 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

\[
\text{I} \\
\rightarrow \text{1}
\]
Strategy II:

- Use 2 Indexes; Manipulate Pointers

Toy → Name=Joe, DEPT=Sales, SAL=15k → Sal > 50k

Strategy III:

- Multiple Key Index

One idea:

Example Record

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:

- <X1, Y1, Attributes>
- <X2, Y2, 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**

- kd-Trees (very similar to what we described here)
- Quad Trees
- R Trees
- ...

**Two more types of multi key indexes**

- Grid
- Partitioned hash

**Grid Index**

![Grid Index Diagram]

**CLAIM**

- Can quickly find records with
  - $key_1 = V_i$ and $key_2 = X_j$
  - $key_1 = V_i$
  - $key_2 = X_j$

- And also ranges....
  - E.g., $key_1 \geq V_i$ and $key_2 < X_j$
• How do we find entry i, j in linear structure?

\[ \text{pos}(i, j) = S + iN + j \]

**Issue:** Cells must be same size, and N must be constant!

**Issue:** Some cells may overflow, some may be sparse...

---

**Solution:** Use Indirection

With indirection:

• Grid can be regular without wasting space
• We do have price of indirection

---

Can also index grid on value ranges

**Grid files**

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

---
Idea: 010110 1110010

Key1 → h1  h2 → Key2

EX:

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

- Find Emp. with Dept. = Sales ∧ Sal=40k

- Find Emp. with Sal=30k
h1(toy) = 0 000 <Fred>
  h1(sales) = 1 001 <Joe><Jan>
  h1(art) = 1 010 <Mary>

h2(10k) = 01 100 <Sally>
  h2(20k) = 11 101 <Tom><Bill>
  h2(30k) = 01 110 <Andy>

  * Find Emp. with Sal=30k

h1(toy) = 0 000 <Fred>
  h1(sales) = 1 001 <Joe><Jan>
  h1(art) = 1 010 <Mary>

h2(10k) = 01 100 <Sally>
  h2(20k) = 11 101 <Tom><Bill>
  h2(30k) = 01 110 <Andy>

  * 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 5

- Skim the following sections:
  - Sections 14.3.6, 14.3.7, 14.3.8
    [Second Ed: 14.6.6, 14.6.7, 14.6.8]
  - Sections 14.4.2, 14.4.3, 14.4.4
    [Second Ed: 14.7.2, 14.7.3, 14.7.4]
- Read the rest

The BIG picture....

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