Hash Indexes

CS 564- Spring 2018

ACKs: Dan Suciu, Jignesh Patel, AnHai Doan
WHAT IS THIS LECTURE ABOUT?

Hash indexes

• Static Hashing
  – what is the I/O cost?
  – problems with static hashing

• Extendible Hashing
  – insertion
  – deletion
HOW TO EVALUATE AN INDEX?

• What access types does it support?
  – e.g. equality search, range search, etc.
• Time to access a record
• Time to insert a record
• Time to delete a record
• How much space does it use?
HASH INDEXES

- efficient for equality search
- not appropriate for range search

- Types of hash indexes:
  - static hashing
  - extendible (dynamic) hashing
A **hash index** is a collection of **buckets**
  
  - bucket = primary page + overflow pages
  - each bucket contains one or more data entries

To find the bucket for each record, we use a hash function \( h \) applied on the search key \( k \)
  
  - \( N = \) number of buckets
  - \( h(k) \mod N = \) bucket in which the data entry belongs

Records with different search key may belong in the same bucket
**STATIC HASHING: EXAMPLE**

**Person** `(name, zipcode, phone)`
- **search key**: zipcode
- **hash function** $h$: last 2 digits

- **bucket 0**
  - (John, 53400, 23218564)
  - (Alice, 54768, 60743111)

- **bucket 1**
  - (Paris, 53409, 23200564)

- **bucket 2**
- **bucket 3**
  - (Maria, 34411, 29010533)

- **4 buckets**
- **each bucket has 2 data entries (full record)**
- **overflow pages**
  - (Anna, 53632, 23209964)
OPERATIONS ON HASH INDEXES

Equality search \((search\text{-}key = value)\)

• apply the hash function on the search key to locate the appropriate bucket

• search through the primary page (plus overflow pages) to find the record(s)

\[ I/O \text{ cost} = 1 + \#\text{overflow pages} \]
OPERATIONS ON HASH INDEXES

• Deletion
  – find the appropriate bucket, delete the record

• Insertion
  – find the appropriate bucket, insert the record
  – if there is no space, create a new overflow page
HASH FUNCTIONS

• An ideal hash function must be uniform: each bucket is assigned the same number of key values
• A bad hash function maps all search key values to the same bucket
• Examples of good hash functions:
  – $h(k) = a * k + b$, where $a$ and $b$ are constants
  – a random function
BUCKET OVERFLOW

• Bucket *overflow* can occur because of
  – insufficient number of buckets
  – *skew* in distribution of records
    • many records have the same search-key value
    • the hash function results in a non-uniform distribution of key values

• Bucket overflow is handled using *overflow buckets*
PROBLEMS OF STATIC HASHING

• In static hashing, there is a **fixed** number of buckets in the index

• Issues with this:
  – if the database grows, the number of buckets will be too small: long overflow chains degrade performance
  – if the database shrinks, space is wasted
  – reorganizing the index is expensive and can block query execution
Extendible Hashing
**Extendible Hashing**

- **Extendible hashing** is a type of *dynamic* hashing
- It keeps a directory of pointers to buckets
- On overflow, it reorganizes the index by **doubling the directory** (and not the number of buckets)
EXTENDIBLE HASHING

To search, use the last 2 digits of the binary form of the search key value.

**local depth**

**global depth**

- 00
- 01
- 10
- 11

2

2

2

2

(John, 12, 23218564)  
(Alice, 8, 60743111)

(Paris, 9, 23200564)

(Maria, 11, 29010533)
EXTENDIBLE HASHING: INSERT

If there is space in the bucket, simply add the record

- (John, 12, 23218564)
- (Alice, 8, 60743111)
- (Paris, 9, 23200564)
- (Zoe, 13, 23345563)
- (Maria, 11, 29010533)
EXTENDIBLE HASHING: INSERT

If the bucket is full, split the bucket and redistribute the entries.

- Global depth increases by 1.
- Local depth increases for the split bucket!
- Local depth remains the same for the other buckets.
EXAMPLE

each page can hold at most two records

We always have: global depth $\geq$ local depth
Example

- The catalog doubles in size
- Global depth becomes 2

Insert: (22,...)

The bucket is split into two buckets with local depth 2

This bucket remains the same
EXAMPLE

There is space in the bucket so nothing changes!

insert: (32,...)

g = 2

00
01
10
11

l=2

(4,...) (32,...)

l=2

(2,...) (22,...)

l=1

(1,...) (7,...)
EXAMPLE

Since local depth is smaller than global, no need to change the directory size!

The bucket is split into two

insert: (3, ...)

\[
g = 2
\]

\[
\begin{array}{c}
(4, \ldots) \\
(32, \ldots)
\end{array}
\]

\[
\begin{array}{c}
(2, \ldots) \\
(22, \ldots)
\end{array}
\]

\[
\begin{array}{c}
(1, \ldots)
\end{array}
\]

\[
\begin{array}{c}
(3, \ldots) \\
(7, \ldots)
\end{array}
\]
EXTENDIBLE HASHING: DELETE

• Locate the bucket of the record and remove it
• If the bucket becomes empty, it can be removed (and update the directory)
• Two buckets can also be coalesced together if the sum of the entries fit in a single bucket
• Decreasing the size of the directory can also be done, but it is expensive
MORE ON EXTENDIBLE HASHING

• How many disk accesses for equality search?
  – One if directory fits in memory, else two

• Directory grows in spurts, and, if the distribution of hash values is skewed, the directory can grow very large

• We may need overflow pages when multiple entries have the same hash value!