COSC2406/2407 Database Systems

File Organisations and Indexing

Xiangmin (Emily) Zhou

Onli

Thursdays

Email : xiangmin.zhou@rm

Lecture 4

References: Ramakrishnan & Gehrke Chapter 8
Garcia-Molina et al. Chapter 13
Elmasri & Navathe Chapters 5 & 6

Slot Offset Table contains of 6 bytes (12 bytes when pagesize $>$ 64KiB) per record:

- 2 bytes page offset for record
- 2 bytes length of record on this page
- 2 byt
  this

Note: 1 KiB (kibibyte) $=$ 1024 bytes
similarly 1 MiB (mibibyte) $=$ 1024^2 bytes
to avoid confusion with 1MB (megabyte)
etc

http://db.apache.org/derby/papers/pageformats.html

In the first part of this lecture, we will:

- Introduce the cost model
- Analyse three common file organisations:
  -
  -
  -

In the second part of this lecture, we will continue wit

indexes.

- Discuss properties of an index
- Discuss alternatives for data entries in an index

Cost Model for Our Analysis

In our discussion we will use a simple cost model.
The cost metric is the number of disk-block I/Os.
Usually the number of I/Os is the dominant cost in database applications. We ignore CP *ccess*).
We expre

- *B*: th
- *D*: (average) time to transfer a disk block

(The average-case analyses here are based on sev assumptions.)

Simplifying Assumptions

Aside from CPU costs and *blocked access*, in our cost model we will ignore:

- tim
- tim

In 2003 th
order of 15 milliseconds. Therefore, I/O is the dom
These trends will continue to diverge: CPU spee
more quickly than disk access speeds — both hav
factor of over 100 since 2003.

We will consider a file that stores data from the following `Character` relation:

NAME	LEVEL	CLASS
Varra	19	
Meerkat	18	
Shaka	2	
Cass	15	
Otho	24	Hunter

The operations we analyse are those identified last lecture:

- Scan:

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file

Basic Operations for File Organisations

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n equ

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n equ "Fi
- Se

Basic Operations for File Organisations

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi

- Se nge
  selection

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi

- Se nge
  selection
  "Find all records of characters with /

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi

- Se nge
  selection
  "Find all records of characters with /

- Insert:

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi

- Se nge
  selection
  "Find all records of characters with /

- Insert: Insert a single, new record into the fi

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n
  equ
  "Fi
- Se nge
  selection
  "Find all records of characters with /
- Insert: Insert a single, new record into the fi
- Delete:

Basic Operations for File Organisations

The operations we analyse are those identified last lecture:

- Scan: Fetch all records in the file
- Se n equ "Fi

- Se nge selection "Find all records of characters with /

- Insert: Insert a single, new record into the fi
- Delete: Delete a single record specified b rid

Example (again)

We will consider a file that stores data from the following `Character` relation:

NAME	LEVEL	CLASS
Varra	19	
Meerkat	18	
Shaka	2	
Cass	15	
Otho	24	Hunter

Records are unordered:

Frost	38	Mage
Shaka	2	
Cass	15	
Otho	24	

Remember that in a heap file, records in the file are unorganised. Here, for simplicity, we assume insertions are always at the end of file. Equality selections are on a unique key, that is, we have exactly one match.

Access costs on average:

- Sca
- Equ
- Range Search: BD
- Insert: $2D$
- Delete: $Search + D$

To ensure a compact heap file, we need to keep and update a free space list for deletions and insertions (using the structures we discussed last week).

Linear Search

We wish to search for a data entry with key value 73 in the following array of 16 items:

Slot	0														14	15
Key	7														92	94

For a linear search, the average cost is:

$$\frac{N}{2} = \frac{16}{2} =$$

Suppose that we again wish to search for a data entry with key value 73 in the following array of 16 items, this time using *binary search*:

Slot	0														14	15
Key	7														92	94

For a binary search, the average cost is:

$$\log_2 N = \log_2 1$$

Records are sorted on name:

Cass	15	Mage
Otho	24	
Shaka	2[illegible]	
Varra	19	

Sorted Files

A sorted file is like a heap file, but the file is sorted on a sequence of fields, which we call the *search key*.

A *search key* need not uniquely identify records. We can locate a record using a binary search on the search key.

I/O cost on

- Sca

- Equality Search: $D \log_2 B$
- Range Search: $D(\log_2 B +$ number of p

- Insert: Search $+BD$
- Delete: Search $+BD$

Inserting and expanding records can be problematic.

Hash function: `level` mod 3

2			

Hash function: `level` mod 3

	Cass	15	Mage
	Frost	38	
2	Moon	20	
	Shaka	2	Shaman

Page Occupancy

Suppose that 100 records are to be stored in a file, and that 10 records can fit on one page.

- How many pages are needed?

Now sup

- How many records fit on one page?

- How many pages are needed in total?

Static Hashed Files

The pages in a hashed file are grouped into buckets. We can apply a hash function to the search key to find out the bucket number to which a record belongs. We assume that we do not have overflow buckets. The page occupancy is assumed to be 80%. I/O cost on average:

- Sca — cords)
- Equ
- Ran
- Insert: $2D$
- Delete: $2D$

Overflowing buckets decrease the performance of

Dynamic hash structures such as *Linear Hashing*, and *Extendible Hashing* address this problem.

Cost of Operation : Summary

Access Method	Heap File	Sorted File	Hashed File
Scan	BD	BD	$1.25BD$
Equalit	0		
Range S			
Insert	2		
Delete	Search $+D$	Search	

No file organisation is uniformly superior in all situatio sed to speed up operations that are not efficiently supporte organisation.

Review: Alternative File Organisations

- Heap files: suitable when the typical access is a file scan to retrieve all records
- Sorted or *sequential files*: best if records must be retrieved in so
- Ha con
  - File is a collection of *buckets*; Bucket = *primary* page plus zero or mo
  - *Hashing function h* maps a record y at some of the fields of *r*, called the

Each file organisation works well for some situations but not for all.

Indexes

An *Index* on a file speeds up selections on the *search key*

- Any subset of the fields of a relation can be the search key of an ind
- A *s* that uni

An index contains a collection of *data ent*
retrieval of all data entries $k*$ with a given s
There are three alternatives for a data entry

Alternatives for the Data Entry $k*$ in an Index

Three alternatives:

1. Data record with search key value k
2. (k, `rid` of data record with search key value k)
3. (k, li

The choi

index tec k. Any

indexing technique can use one of the three altern

Examples of indexing techniques include B+-tr

structures.

Typically, an index contains auxiliary informati

to the desired data entries (for example, index entries in index pages in a B+-tree).

- Alternative 1:
  - If this is used, the index structure is in fact a file organisation for
  -
  - If data records are very large, the number of entries is high. This typically implies that th information in the index is also large.

An Alternative 1 Index

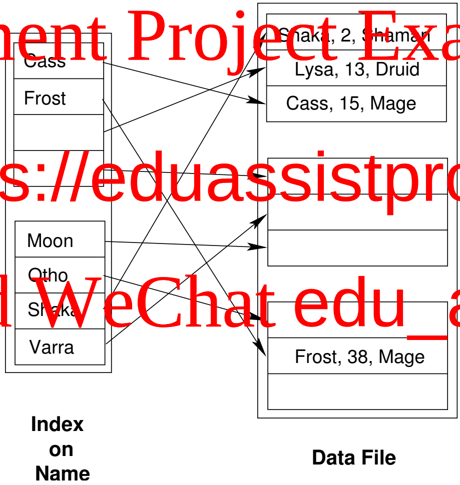
An Alternative 1 index on `level`:

Shaka	2	Shaman
Moon	20	
Otho	24	
Frost	38	

- Alternatives 2 and 3:
  -
  -
  -
  - Alternative 3 is most compact, but the vari entries is harder to handle (lists can grow a

Alternative 2 Index

An Alternative 2 index on `name`:



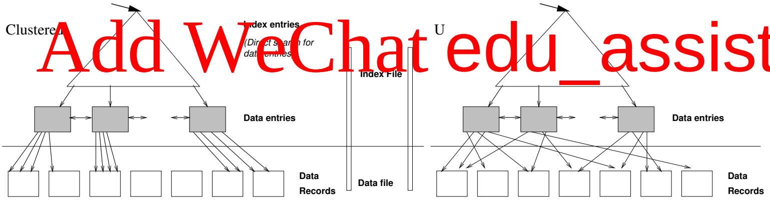
Index Classification

- *Primary* vs. *secondary*: If the search key contains the primary key, then the index is the primary index.
  -
- *Clu*
  or “cl
  clu

  - Using alternative 1 implies a clustered ind
  - A file can be clustered at most on one search
  - The cost of retrieving data records greatly
    index is clustered or not

Consider using alternative 2 used for the data entries and storing the data records in a heap file.

- To build clustered index, first sort the heap file (leaving free space on each page for future inserts).
- Ove
  'clo

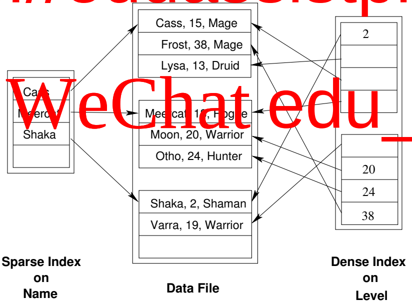


- *Dense* vs *sparse*: If there is (at least) one data entry in the index per search key value then the index is dense. In a sparse index, we may have one data entry in the index for a page or set of rec
- Imp
  -
  -
    order)
  - There is only one sparse index (since we ca clustered index)
  - Sparse indexes are smaller; however, s are based on dense indexes (refer to Section 12.5.2 of Ramakrishnan & Gehrke)

Dense vs. Sparse Indexes ...

The first index shown below is a sparse, clustered index on *name*. The order of data entries in the index corresponds to the order of records in the data file. There is one data entry per page of records.

The second index is dense, unclustered index on *level*. The order of data entries in the index differs from the order of data records. (There is one data entry in the i



Advantages and Disadvantages

- Clustered index: good for range queries. *Rids* of qualifying index entries point to a contiguous collection of records, hence few page I/Os
- Un
  mat
  add
- De
  memory; can find a record with one I/O. Can d
  alone whether a record exists
- Sparse index: smaller than a dense index, s
  memory and can be searched quickly. However, may need to to an I/O just to check whether a record exists

Review

- Many alternative file organisations exist, each is appropriate for particular situations
- If sel
  ind

  -
  -
    and also good for equality searches
- An index is a collection of data entries, plus a w

  entries with given key values

- Index data entries can be actual data records, (key, `rid`) pairs, or (key, `rid`-list) pairs.
  -

- The
  wit
- Indexes can be classified as clustered or un
  secondary, and dense or sparse. Differen
  consequences for utility and performanc

Summary: File Organisations and Properties of Indexes

We have discussed:

- A file-access cost model based on the number of disk page I/Os as the cost metric.
- Thr
- the p

    -
    - *dense* vs. *sparse*;
    - *primary* vs. *secondary*.

- Alternatives for the index data entries *k*

In the next few lectures, we will cover hash-based indexing and tree-based indexing techniques like the B+ tree. We will also discuss a related topic, the external merge sort.