CS 525: Advanced Database Organization
03: Disk Organization

Boris Glavic

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

Topics for today

- How to lay out data on disk
- How to move it to/from memory

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

⟹ What we have available: Bytes

← 8 →
bits

To represent:

- Integer (short): 2 bytes
  e.g., 35 is

  | 00000000 | 00100011 |

  Endian! Could as well be

  | 00100011 | 00000000 |

- Real, floating point
  n bits for mantissa, m for exponent….

To represent:

- Characters
  → various coding schemes suggested,
    most popular is ASCII (1 byte encoding)

  Example:
  A: 1000001
  a: 1100001
  5:   0110101
  LF: 0001010

To represent:

- Boolean
  e.g., TRUE   `1111 1111`
        FALSE   `0000 0000`
- Application specific
      e.g., enumeration
            RED $\rightarrow$ 1   GREEN $\rightarrow$ 3
            BLUE $\rightarrow$ 2   YELLOW $\rightarrow$ 4 …

CS 525   Notes 3   7   IIT College of Science and Letters

To represent:

- Boolean
  e.g., TRUE   `1111 1111`
        FALSE   `0000 0000`
- Application specific
      e.g., RED $\rightarrow$ 1   GREEN $\rightarrow$ 3
            BLUE $\rightarrow$ 2   YELLOW $\rightarrow$ 4 …

$\Rightarrow$ Can we use less than 1 byte/code?
      Yes, but only if desperate…

CS 525   Notes 3   8   IIT College of Science and Letters

To represent:

- Dates
      e.g.: - Integer, # days since Jan 1, 1900
           - 8 characters, YYYYMMDD
           - 7 characters, YYYYDDD
                (not YYMMDD! Why?)
- Time
      e.g. - Integer, seconds since midnight
         - characters, HHMMSSFF

CS 525   Notes 3   9   IIT College of Science and Letters

To represent:

- String of characters
  – Null terminated
      e.g., | c | a | t | ⊠ | |

  – Length given
      e.g., | 3 | c | a | t | ⊠ | |

  - Fixed length

CS 525   Notes 3   10   IIT College of Science and Letters

To represent:

- Bag of bits

  | Length | Bits |

CS 525   Notes 3   11   IIT College of Science and Letters

Key Point

- Fixed length items

- Variable length items
      - usually length given at beginning

CS 525   Notes 3   12   IIT College of Science and Letters

Also

- Type of an item: Tells us how to
                     interpret
                     (plus size if fixed)

Overview   Data Items
                       ↓
                    Records
                       ↓
                    Blocks
                       ↓
                    Files
                       ⋮
                    Memory

Record - Collection of related data items (called FIELDS)

E.g.: Employee record:
           name field,
           salary field,
           date-of-hire field, ...

Types of records:

- Main choices:
  – FIXED vs VARIABLE FORMAT
  – FIXED vs VARIABLE LENGTH

Fixed format

A SCHEMA (not record) contains
    following information
      - # fields
      - type of each field
      - order in record
      - meaning of each field

Example: fixed format and length

Employee record
    (1) E#, 2 byte integer
    (2) E.name, 10 char.   Schema
    (3) Dept, 2 byte code

    | 55 | s m i t h | | 02 |
                                       Records
    | 83 | j o n e s | | 01 |

Variable format

- Record itself contains format "Self Describing"

Example: variable format and length

2	5	I	46	4	S	4	F	O	R	D

- # Fields
- Code identifying field as E#
- Integer type
- Code for Ename
- String type
- Length of str.
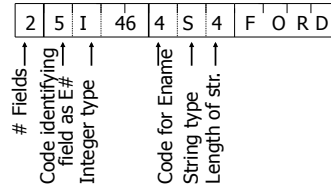
Field name codes could also be strings, i.e. TAGS

Variable format useful for:

- "sparse" records
- repeating fields
- evolving formats

········→ But may waste space...

- <u>EXAMPLE:</u> var format record with repeating fields

  Employee → one or more → children

3	E_name: Fred	Child: Sally	Child: Tom

<u>Note:</u> Repeating fields does not imply
- variable format, nor
- variable size

John	Sailing	Chess	--

<u>Note:</u> Repeating fields does not imply
- variable format, nor
- variable size

John	Sailing	Chess	--

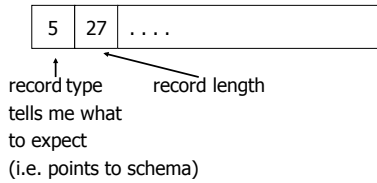- Key is to allocate maximum number of repeating fields (if not used → null)

Slide 25

☆ Many variants between fixed - variable format:

<u>Example:</u> Include <u>record type</u> in record

5	27	. . . .

record type
tells me what
to expect
(i.e. points to schema)

record length

Slide 26

<u>Record header</u> - data at beginning that describes record

May contain:
- record type
- record length
- time stamp
- null-value bitmap
- other stuff ...

Slide 27

<u>Other interesting issues:</u>

- Compression
  - within record - e.g. code selection
  - collection of records - e.g. find common patterns
- Encryption
- Splitting of large records
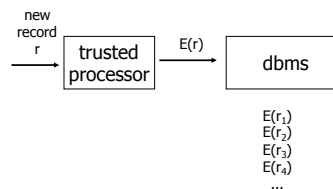 - E.g., image field, store pointer

Slide 28

Record Header – null-map

- SQL: NULL is special value for every data type
  - Reserve one value for each data type as NULL?
- Easier solution
  - Record header has a bitmap to store whether field is NULL
  - Only store non-NULL fields in record
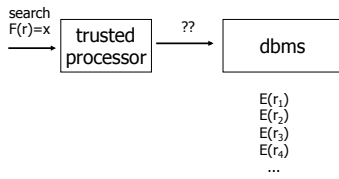
Slide 29

Separate Storage of Large Values

- Store fields with large values separately
  - E.g., image or binary document
  - Records have pointers to large field content
- Rationale
  - Large fields mostly not used in search conditions
  - Benefit from smaller records

Slide 30
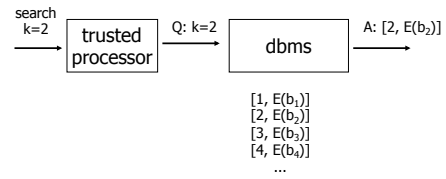
Encrypting Records

new record r → trusted processor → E(r) → dbms

$E(r_1)$
$E(r_2)$
$E(r_3)$
$E(r_4)$
...

Encrypting Records

search
$F(r)=x$ → trusted processor → ?? → dbms

$E(r_1)$
$E(r_2)$
$E(r_3)$
$E(r_4)$
...

CS 525   Notes 3   31   IIT College of Science and Letters

Search Key in the Clear

search
$k=2$ → trusted processor → Q: $k=2$ → dbms → A: $[2, E(b_2)]$

$[1, E(b_1)]$
$[2, E(b_2)]$
$[3, E(b_3)]$
$[4, E(b_4)]$
...

- each record is [k,b]
- store [k, E(b)]
- can search for records with k=x

CS 525   Notes 3   32   IIT College of Science and Letters

Encrypt Key

search
$k=2$ → trusted processor → Q: $k' = E(2)$ → dbms → A: $[E(2), E(b_2)]$

$[E(1), E(b_1)]$
$[E(2), E(b_2)]$
$[E(3), E(b_3)]$
$[E(4), E(b_4)]$
...
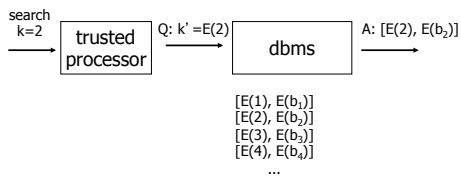
- each record is [k,b]
- store [E(k), E(b)]
- can search for records with k=E(x)

CS 525   Notes 3   33   IIT College of Science and Letters

Issues

- Hard to do range queries
- Encryption not good
- Better to use encryption that does not always generate same cyphertext

k → E → $E(k, random)$ → D → k
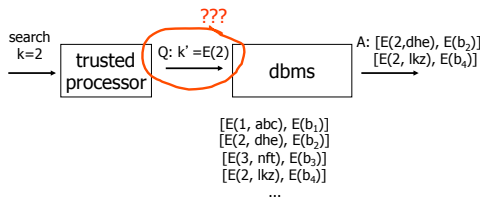
simplification
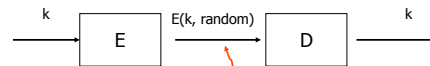
CS 525   Notes 3   34   IIT College of Science and Letters

How Do We Search Now?

???

search
$k=2$ → trusted processor → Q: $k' = E(2)$ → dbms → A: $[E(2,dhe), E(b_2)]$
$[E(2, lkz), E(b_4)]$

$[E(1, abc), E(b_1)]$
$[E(2, dhe), E(b_2)]$
$[E(3, nft), E(b_3)]$
$[E(2, lkz), E(b_4)]$
...

- each record is [k,b]
- store [E(k, rand), E(b)]
- can search for records with k=E(x,???)?

CS 525   Notes 3   35   IIT College of Science and Letters

Solution?

- Develop new decryption function:
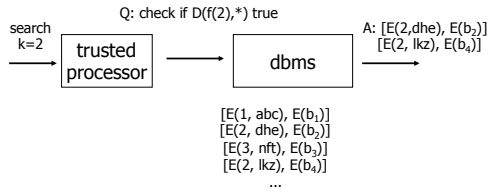 $D(f(k_1), E(k_2, rand))$ is true if $k_1=k_2$

CS 525   Notes 3   36   IIT College of Science and Letters

Solution?

- Develop new decryption function:
  $D(f(k_1), E(k_2, rand))$ is true if $k_1 = k_2$

Q: check if $D(f(2),*)$ true

search
k=2 → trusted processor → dbms → A: [E(2,dhe), E(b_2)]
[E(2, lkz), E(b_4)]

[E(1, abc), E(b_1)]
[E(2, dhe), E(b_2)]
[E(3, nft), E(b_3)]
[E(2, lkz), E(b_4)]
...

Issues?

- Cannot do non-equality predicates
- Hard to build indexes

<u>Next:</u> placing records into blocks

blocks [ ] [ ] ... [ ]

a file

<u>Next:</u> placing records into blocks

assume fixed length blocks

blocks [ ] [ ] ... [ ]

a file ← assume a single file (for now)

Options for storing records in blocks:

(1) separating records
(2) spanned vs. unspanned
(3) sequencing
(4) indirection

---

(1) Separating records

Block   [R1 | R2 | R3]
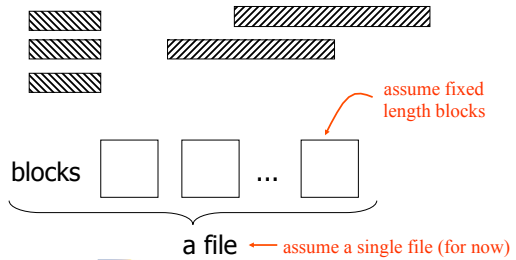
(a) no need to separate - fixed size recs.
(b) special marker
(c) give record lengths (or offsets)
    - within each record
    - in block header

(2) Spanned vs. Unspanned

- Unspanned: records must be within one block
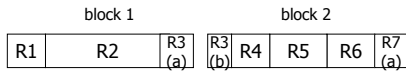
block 1			block 2			
R1	R2	▨	R3	R4	R5	▨	...

- Spanned

block 1			block 2				
R1	R2	R3 (a)	R3 (b)	R4	R5	R6	R7 (a)
...

With spanned records:
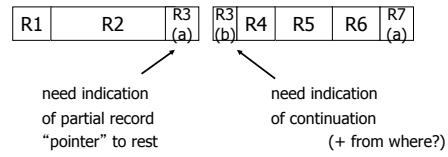
R1	R2	R3 (a)	R3 (b)	R4	R5	R6	R7 (a)

need indication   need indication
of partial record   of continuation
"pointer" to rest   (+ from where?)

Spanned vs. unspanned:

- Unspanned is <u>much</u> simpler, but may waste space…
- Spanned essential if
    record size > block size

---

(3) Sequencing

- Ordering records in file (and block) by some key value

  <u>Sequential file</u> ($\Rightarrow$ sequenced)

<u>Why sequencing?</u>

Typically to make it possible to efficiently read records in order

(e.g., to do a merge-join — discussed later)

<u>Sequencing Options</u>

(a) Next record physically contiguous
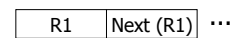
R1	Next (R1)	...

(b) Linked

R1		Next (R1)	

Sequencing Options

(c)   Overflow area

Records
in sequence

R1
R2
R3
R4
R5

Sequencing Options

(c)   Overflow area

Records
in sequence

header	
R1	
R2	
R3	
R4	
R5	

R2.1
R1.3
R4.7

(4) Indirection

• How does one refer to records?

⟶ | Rx |

(4) Indirection

• How does one refer to records?

⟶ | Rx |
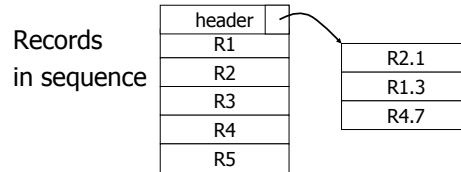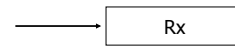
Many options:
  Physical ⟷ Indirect

☆ Purely Physical
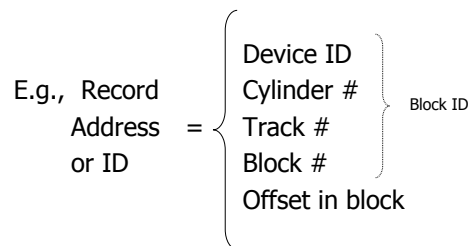
E.g., Record
       Address   =
       or ID
       { Device ID
         Cylinder #
         Track #   } Block ID
         Block #
         Offset in block }

☆ Fully Indirect

E.g., Record ID is arbitrary bit string
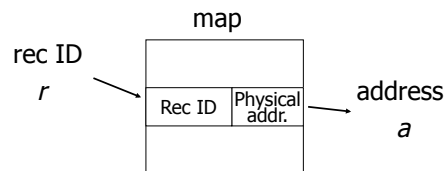
map

rec ID
r

Rec ID	Physical addr.

address
a

9

Tradeoff

Flexibility ⟷ Cost

to move records   of indirection

(for deletions, insertions)

Physical ⟷ Indirect

↑

Many options
in between …

Example: Indirection in block
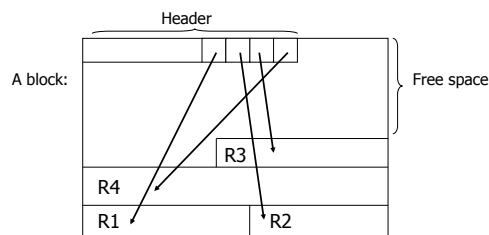
Tuple Identifier (TID)

- TID is
  - Page identifier
  - Slot number
- Slot stores either record or pointer (TID)
- TID of a record is fixed for all time

TID Operations

- Insertion
  - Set TID to record location (page, slot)
- Moving record
  - e.g., update variable-size or reorganization
  - Case 1: TID point to record
    - Replace record with pointer (new TID)
  - Case 2: TID points to pointer (TID)
    - Replace pointer with new pointer

TID Properties

- TID of record never changes
  - Can be used safely as pointer to record (e.g., in index)
- At most one level of indirection
  - Relatively efficient
  - Changes to physical address - changing max 2 pages

Slide 61

<u>Block header</u> - data at beginning that describes block

May contain:
- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, …)
- Pointer to other blocks "like it"
- Timestamp ...

Slide 62

Options for storing records in blocks:

(1) separating records
(2) spanned vs. unspanned
(3) sequencing
(4) indirection

Slide 63
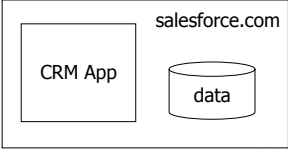
Case Study: salesforce.com

- salesforce.com provides CRM services
- salesforce customers are *tenants*
- Tenants run apps and DBMS as service

tenant A

tenant B

tenant C

salesforce.com

CRM App   data

Slide 64

Options for Hosting

- Separate DBMS per tenant
- One DBMS, separate tables per tenant
- One DBMS, shared tables

Slide 65

Tenants have similar data

tenant 1:

customer	A	B	C	D	E	F
	a1	b1	c1	d1	e1	-
	a2	b2	c2	-	e2	f2

tenant 2:

customer	A	B	C	D	G
	a3	b3	c2	-	-
	a1	b1	c1	-	g1
	a4	-	-	d1	

Slide 66

salesforce.com solution

customer	tenant	A	B	C
	1	a1	b1	c1
	1	a2	b2	c2
	2	a3	b3	c2
	2	a1	b1	c1

← fixed schema for all tenants

cust-other	tenant	A	f1	v1	f2	v2 ...
	1	a1	D	d1	E	e1
	1	a2	E	e2	F	f2
	2	a1	G	g1		
	3	a4	D	d1		

← var schema for all tenants
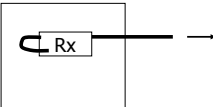
Other Topics

(1) Insertion/Deletion
(2) Buffer Management
(3) Comparison of Schemes

Deletion

Block

Options:

(a)   Immediately reclaim space
(b)   Mark deleted

Options:

(a)   Immediately reclaim space
(b)   Mark deleted
  – May need chain of deleted records
    (for re-use)
  – Need a way to mark:
    • special characters
    • delete field
    • in map

☆ As usual, many tradeoffs...

• How expensive is to move valid record to free space for immediate reclaim?
• How much space is wasted?
  – e.g., deleted records, delete fields, free space chains,...

Concern with deletions

Dangling pointers

Solution #1: Do not worry
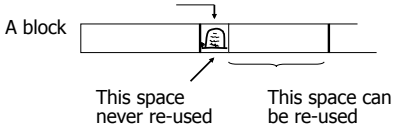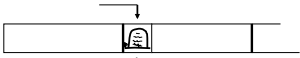
Solution #2: Tombstones

E.g., Leave "MARK" in map or old location

Solution #2: Tombstones

E.g., Leave "MARK" in map or old location
- Physical IDs

A block
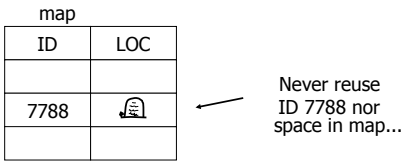
This space never re-used   This space can be re-used

Solution #2: Tombstones

E.g., Leave "MARK" in map or old location

- Logical IDs

map

ID	LOC
7788	

Never reuse ID 7788 nor space in map...

Insert

Easy case: records not in sequence
  → Insert new record at end of file or in deleted slot
  → If records are variable size, not as easy...

Insert

Hard case: records in sequence
  → If free space "close by", not too bad...
  → Or use overflow idea...

Interesting problems:

- How much free space to leave in each block, track, cylinder?
- How often do I reorganize file + overflow?

Free space
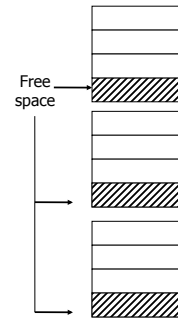

Buffer Management

- DB features needed
- Buffer Replacement Strategies
  - E.g., LRU, clock
- Pinned blocks
- Forced output - - - - - - - → in Notes02
- Double buffering
- Swizzling

Buffer Manager

- Manages blocks cached from disk in main memory
- Usually -> fixed size buffer (M pages)
- DB requests page from Buffer Manager
  - Case 1: page is in memory -> return address
  - Case 2: page is on disk -> load into memory, return address

---

Goals

- Reduce the amount of I/O
- Maximize the *hit rate*
  - Ratio of number of page accesses that are fulfilled without reading from disk
- -> Need strategy to decide when to

Buffer Manager Organization

- Bookkeeping
  - Need to map (hash table) page-ids to locations in buffer (**page frames**)
  - Per page store *fix count, dirty bit, …*
  - Manage free space
- Replacement strategy
  - If page is requested but buffer is full
  - Which page to emit remove from buffer

FIFO

- **F**irst **I**n, **F**irst **O**ut
- Replace page that has been in the buffer for the longest time
- Implementation: E.g., pointer to oldest page (circular buffer)
  – Pointer->next = Pointer++ % M
- Simple, but not prioritizing frequently accessed pages

LRU

- Least Recently Used
- Replace page that has not been accessed for the longest time
- Implementation:
  – List, ordered by LRU
  – Access a page, move it to list tail
- Widely applied and reasonable performance

Clock

- Frames are organized clock-wise
- Pointer S to current frame
- Each frame has a reference bit
  – Page is loaded or accessed -> bit = 1
- Find page to replace (advance pointer)
  – Return first frame with bit = 0
  – On the way set all bits to 0

Clock Example

Reference bit

S →	0	Page 0
	1	Page 1
	1	Page 2
	0	Page 3
	1	Page 4

Other Replacement Strategies

- LRU-K
- GCLOCK
- Clock-Pro
- ARC
- LFU

Swizzling

Memory   Disk

block 1   block 1

Rec A   block 2

Swizzling

Memory | Disk

block 1

block 2 — Rec_A

block 1

block 2 — Rec_A
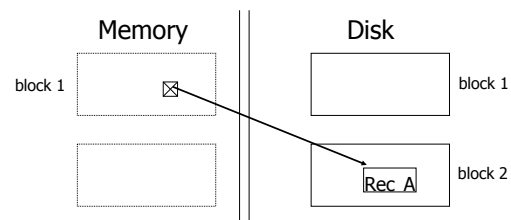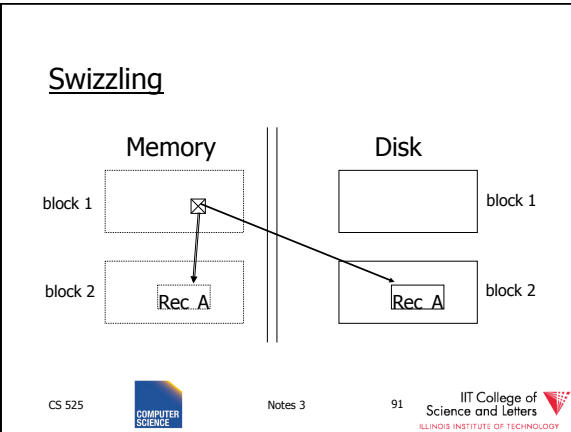
Row vs Column Store

- So far we assumed that fields of a record are stored contiguously (<u>row store</u>)...
- Another option is to store like fields together (<u>column store</u>)

Row Store

- Example: Order consists of
  - id, cust, prod, store, price, date, qty

| id1 | cust1 | prod1 | store1 | price1 | date1 | qty1 |

| id2 | cust2 | prod2 | store2 | price2 | date2 | qty2 |

| id3 | cust3 | prod3 | store3 | price3 | date3 | qty3 |

Column Store

- Example: Order consists of
  - id, cust, prod, store, price, date, qty

id1	cust1		id1	prod1		id1	price1	qty1
id2	cust2		id2	prod2		id2	price2	qty2
id3	cust3		id3	prod3		id3	price3	qty3
id4	cust4		id4	prod4		id4	price4	qty4
...	...		...	...		...	...	...

ids may or may not be stored explicitly

Row vs Column Store

- Advantages of Column Store
  - more compact storage (fields need not start at byte boundaries)
  - efficient reads on data mining operations
- Advantages of Row Store
  - writes (multiple fields of one record)more efficient
  - efficient reads for record access (OLTP)

Comparison

- There are 10,000,000 ways to organize my data on disk...

  Which is right for me?

Issues:

Flexibility ———— Space Utilization

Complexity ———— Performance

(Flexibility and Performance crossed with Space Utilization and Complexity)

☆ To evaluate a given strategy, compute following parameters:
- -> space used for expected data
- -> expected time to
  - fetch record given key
  - fetch record with next key
  - insert record
  - append record
  - delete record
  - update record
  - read all file
  - reorganize file

Example

How would you design Megatron 3000 storage system? (for a relational DB, low end)
- – Variable length records?
- – Spanned?
- – What data types?
- – Fixed format?
- – Record IDs ?
- – Sequencing?
- – How to handle deletions?

Summary

- How to lay out data on disk

Data Items
↓
Records
↓
Blocks
↓
Files
↓
Memory
↓
DBMS

Next

How to find a record quickly, given a key