

CS 277: Database System Implementation

Notes 03: Disk Organization

Arthur Keller

CS 277 – Spring 2002

Notes 3

1

Topics for today

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

CS 277 – Spring 2002

Notes 3

2

What are the data items we want to store?

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

⇒ What we have available: Bytes



CS 277 – Spring 2002

Notes 3

3

To represent:

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

00000000 00100011

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

CS 277 – Spring 2002

Notes 3

4

To represent:

- Characters
→ various coding schemes suggested,
most popular is ASCII

Example:

A: 1000001

a: 1100001

5: 0110101

LF: 0001010 (line feed)

CS 277 – Spring 2002

Notes 3

5

To represent:

- Boolean

e.g., TRUE 1111 1111

FALSE 0000 0000

- Application specific
e.g., RED → 1 GREEN → 3
BLUE → 2 YELLOW → 4 ...

⇒ Can we use less than 1 byte/code?

Yes, but only if desperate...

CS 277 – Spring 2002

Notes 3

6

To represent:

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

To represent:

- String of characters
 - Null terminated
 - e.g.,

c	a	t	⊗		
---	---	---	---	--	--
 - Length given
 - e.g.,

3	c	a	t	⊗	
---	---	---	---	---	--
 - Fixed length

To represent:

- Bag of bits



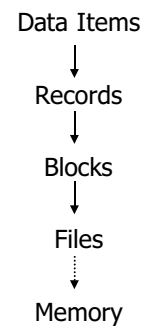
Key Point

- Fixed length items
- Variable length items
 - usually length given at beginning

Also

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

Overview



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

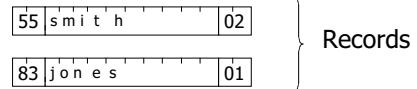
A SCHEMA (not record) contains following information

- # fields
- type of each field
- order in record
- meaning of each field

Example: fixed format and length

Employee record

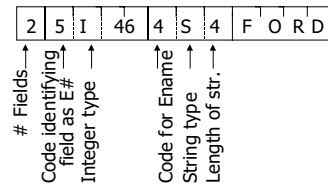
- (1) E#, 2 byte integer
 - (2) E.name, 10 char.
 - (3) Dept, 2 byte code
- } Schema



Variable format

- Record itself contains format "Self Describing"

Example: variable format and length



Field name codes could also be strings, i.e. TAGS

Variable format useful for:

- "sparse" records
- repeating fields
- evolving formats

-----> But may waste space...

- EXAMPLE: variable format record with repeating fields
Employee → one or more → children

3	E_name: Fred	Child: Sally	Child: Tom
---	--------------	--------------	------------

Note: 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)

☆ Many variants between fixed - variable format:

Ex. #1: Include record type in record

5	27
---	----	------

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

Record header - data at beginning that describes record

May contain:

- record type
- record length
- time stamp
- other stuff ...

Ex #2 of variant between FIXED/VAR format

- Hybrid format
- one part is fixed, other variable

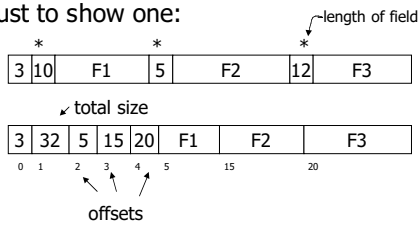
E.g.: All employees have E#, name, dept other fields vary.

25	Smith	Toy	2	Hobby:chess	retired
----	-------	-----	---	-------------	---------

↑ # of var fields

☆ Also, many variations in internal organization of record

Just to show one:



Question:

We have seen examples for

- * Fixed format and length records
- * Variable format and length records

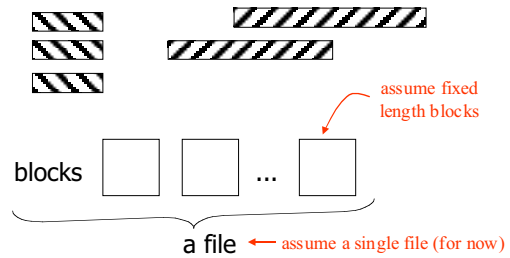
(a) Does fixed format and variable length make sense?

(b) Does variable format and fixed length make sense?

Other interesting issues:

- Compression
 - within record - e.g. code selection
 - collection of records - e.g. find common patterns
- Encryption

Next: placing records into blocks



Options for storing records in blocks:

- (1) separating records
- (2) spanned vs. unspanned
- (3) mixed record types - clustering
- (4) split records
- (5) sequencing
- (6) indirection

(1) Separating records



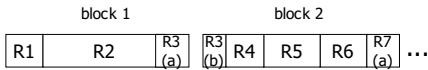
- (a) no need to separate - fixed size recs.
- (b) special marker
- (c) give record lengths (or offsets)
 - within each record
 - in block header

(2) Spanned vs. Unspanned

- Unspanned: records must be within one block



- Spanned

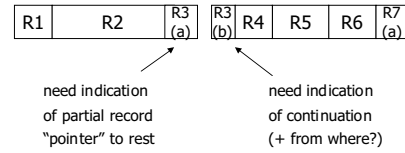


CS 277 – Spring 2002

Notes 3

31

With spanned records:



CS 277 – Spring 2002

Notes 3

32

Spanned vs. unspanned:

- Unspanned is much simpler, but may waste space...
- Spanned essential if record size > block size

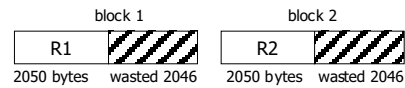
CS 277 – Spring 2002

Notes 3

33

Example

10^6 records
each of size 2,050 bytes (fixed)
block size = 4096 bytes



- Total wasted = 2×10^9 Utiliz. = 50%
- Total space = 4×10^9

CS 277 – Spring 2002

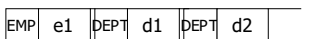
Notes 3

34

(3) Mixed record types

- Mixed - records of different types (e.g. EMPLOYEE, DEPT) allowed in same block

e.g., a block



CS 277 – Spring 2002

Notes 3

35

Why do we want to mix?

Answer: CLUSTERING

Records that are frequently accessed together should be in the same block

CS 277 – Spring 2002

Notes 3

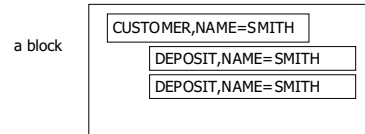
36

Compromise:

No mixing, but keep related records in same cylinder ...

Example

Q1: select A#, C_NAME, C_CITY, ...
from DEPOSIT, CUSTOMER
where DEPOSIT.C_NAME =
CUSTOMER.C.NAME



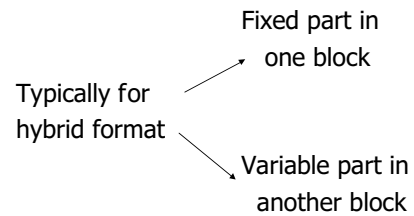
- If Q1 frequent, clustering good
- But if Q2 frequent

Q2: SELECT *

FROM CUSTOMER

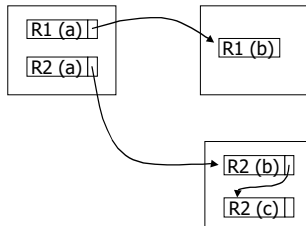
CLUSTERING IS COUNTER PRODUCTIVE

(4) Split records



Block with fixed recs.

Block with variable recs.



This block also has fixed recs.

Question

What is difference between

- Split records
- Simply using two different record types?

(5) Sequencing

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

Sequential file (\Rightarrow sequenced)

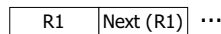
Why sequencing?

Typically to make it possible to efficiently read records in order

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

Sequencing Options

(a) Next record physically contiguous



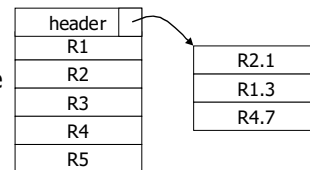
(b) Linked



Sequencing Options

(c) Overflow area

Records in sequence



(6) Indirection

- How does one refer to records?



Many options:

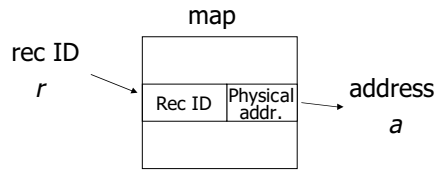
Physical \longleftrightarrow Indirect

☆ Purely Physical

E.g., Record Address or ID = $\left\{ \begin{array}{l} \text{Device ID} \\ \text{Cylinder \#} \\ \text{Track \#} \\ \text{Block \#} \\ \text{Offset in block} \end{array} \right\}$ Block ID

☆ Fully Indirect

E.g., Record ID is arbitrary bit string



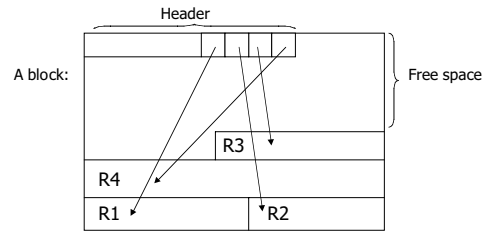
Tradeoff

Flexibility ↔ Cost
 to move records ↔ of indirection
 (for deletions, insertions)

Physical ↔ Indirect

↑
 Many options
 in between ...

Ex #1 Indirection in block



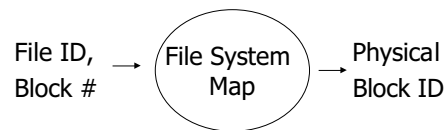
Block header - data at beginning that describes block

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 ...

Ex. #2 Use logical block #'s understood by file system

REC ID → File ID
 Block #
 Record # or Offset



File system map may be "Semi-physical"...

File F1: physical address of block 1
table of bad blocks:

{ B57 → XXX
 B107 → YYY

Rest can be computed via formula...

CS 277 – Spring 2002

Notes 3

55

Num. Blocks: 20
Start Block: 1000
Block Size: 100
Bad Blocks:
 3 → 20,000
 7 → 15,000

Where is Block # 2?

Where is Block # 3?

File DEFINITION

CS 277 – Spring 2002

Notes 3

56

Options for storing records in blocks

- (1) Separating records
- (2) Spanned vs. Unspanned
- (3) Mixed record types - Clustering
- (4) Split records
- (5) Sequencing
- (6) Indirection

CS 277 – Spring 2002

Notes 3

57

Other Topics

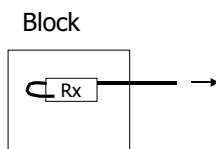
- (1) Insertion/Deletion
- (2) Buffer Management
- (3) Comparison of Schemes

CS 277 – Spring 2002

Notes 3

58

Deletion



CS 277 – Spring 2002

Notes 3

59

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

CS 277 – Spring 2002

Notes 3

60

☆ 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

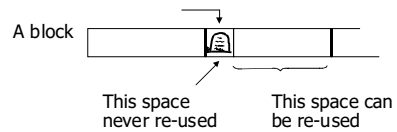


Solution #1: Do not worry

Solution #2: Tombstones

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

- Physical IDs



Solution #2: Tombstones

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

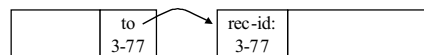
- Logical IDs

map	
ID	LOC
7788	

Never reuse ID 7788 nor space in map...

Solution #3 (?):

- Place record ID within every record
- When you follow a pointer, check if it leads to correct record

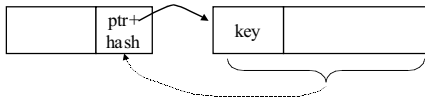


Does this work???

If space reused, won't new record have same ID?

Solution #4 (?):

- To point, use (pointer + hash)
or (pointer + key)?



- What if record modified???

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...

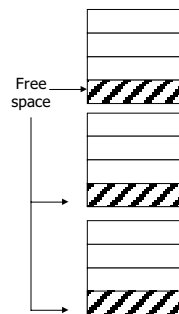
Insert

Hard case: records in sequence

- If free space "close by", not too bad...
- Or use overflow idea...

Interesting problems:

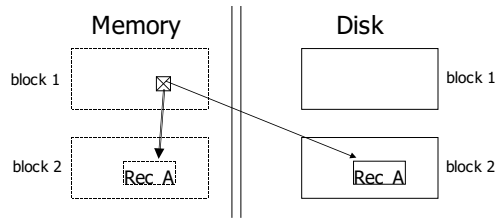
- How much free space to leave in each
block, track, cylinder?
- How often do I reorganize file + overflow?



Buffer Management

- DB features needed
 - Why LRU may be bad
 - Pinned blocks
 - Forced output
 - Double buffering
 - Swizzling
- Read
Textbook!
- in Notes02

Swizzling



CS 277 - Spring 2002

Notes 3

73

One Option:

Translation
Table

DB Addr	Mem Addr
Rec-A	Rec-A-inMem

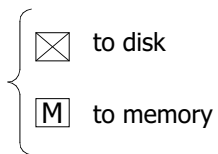
CS 277 - Spring 2002

Notes 3

74

Another Option:

In memory pointers - need "type" bit



CS 277 - Spring 2002

Notes 3

75

Swizzling

- Automatic
- On-demand
- No swizzling / program control

CS 277 - Spring 2002

Notes 3

76

Comparison

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

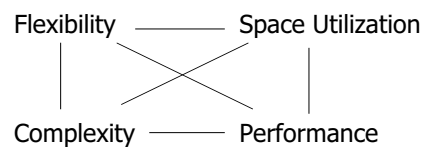
Which is right for me?

CS 277 - Spring 2002

Notes 3

77

Issues:



CS 277 - Spring 2002

Notes 3

78

☆ 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

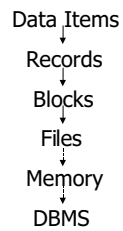
Example

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



Next

How to find a record quickly,
given a key