CS 232A: Database System Principles

Notes 03: Disk Organization

Topics for today

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

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
   \[ \begin{array}{c}
     00000000 \\
     00100011
   \end{array} \]

• Real, floating point
  \( n \) bits for mantissa, \( m \) for exponent....

---

To represent:

• Characters
  \[ \text{various coding schemes suggested, most popular is ascii} \]
  
  Example:
  
  A: 1000001
  a: 1100001
  5: 0110101
  LF: 0001010

---

To represent:

• Boolean
  \[ \begin{array}{c}
     1111 1111 \\
     0000 0000
   \end{array} \]

• 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...
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

To represent:

- **String of characters**
  - Null terminated
    - e.g.,  
  - Length given
    - e.g.,  
  - Fixed length

To represent:

- **Bag of bits**

<table>
<thead>
<tr>
<th>Length</th>
<th>Bits</th>
</tr>
</thead>
</table>
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

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.
(3) Dept, 2 byte code

Schema

Records

Example: variable format and length

Variable format

- Record itself contains format
  “Self Describing”

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: var format record with repeating fields
Employee → one or more → children

[Table]

<table>
<thead>
<tr>
<th>#</th>
<th>E_name</th>
<th>Child:</th>
<th>Child:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Fred</td>
<td>Sally</td>
<td>Tom</td>
</tr>
</tbody>
</table>

Note: Repeating fields does not imply
- variable format, nor
- variable size

- 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

<table>
<thead>
<tr>
<th>record type</th>
<th>record length</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>27</td>
</tr>
</tbody>
</table>

... (i.e. points to schema)

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.

<table>
<thead>
<tr>
<th>Smith</th>
<th>Toy</th>
<th>2</th>
<th>Hobby: chess</th>
<th>retired</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# of var fields
Also, many variations in internal organization of record

Just to show one:

<table>
<thead>
<tr>
<th>F3</th>
<th>F2</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

total size

<table>
<thead>
<tr>
<th>F3</th>
<th>F2</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>

offsets

---

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

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
    - R1
    - R2
  - Block 2
    - ... R3
    - R4
    - R5

- **Spanned**
  - Block 1
    - R1
    - R2
    - R3
  - Block 2
    - R4
    - R5
    - R6

With spanned records:

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

Spanned vs. unspanned:

- Unspanned is much simpler, but may waste space...
- Spanned essential if record size > block size
Example
10⁶ records
each of size 2,050 bytes (fixed)
block size = 4096 bytes

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>2050 bytes</td>
<td>wasted 2046</td>
<td>2050 bytes</td>
</tr>
</tbody>
</table>

- Total wasted = 2 x 10⁹ Utiliz = 50%
- Total space = 4 x 10⁹

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

  e.g., a block

  EMP e1 REP d1 REP d2

Why do we want to mix?
Answer: CLUSTERING

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

Fixed part in one block
Typically for hybrid format
Variable part in another block

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 ( ⇒ 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

<table>
<thead>
<tr>
<th>Records in sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
</tr>
</tbody>
</table>

R1.3
R2.1
R4.7

(6) 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 #, Offset in block \} Block ID


Fully Indirect
E.g., Record ID is arbitrary bit string

Tradeoff
Flexibility \rightarrow Cost
to move records \quad of indirection
(for deletions, insertions)

Physical \rightarrow Indirect
↑
Many options
in between ...
Ex #1 Indirection in block

A block:

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
</tbody>
</table>

Free space

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 ID, Block # → File Syst. Map
    Physical Block ID
File system map may be “Semi-physical”...

File F1: physical address of block 1
table of bad blocks:
\[
\begin{align*}
B57 & \rightarrow XXX \\
B107 & \rightarrow YYY
\end{align*}
\]

Rest can be computed via formula...

Num. Blocks: 20
Start Block: 1000
Block Size: 100
Bad Blocks:
\[
\begin{align*}
3 & \rightarrow 20,000 \\
7 & \rightarrow 15,000
\end{align*}
\]

File DEFINITION

Where is Block # 2?
Where is Block # 3?

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
Other Topics

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

---

Deletion

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,...
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

<table>
<thead>
<tr>
<th>ID</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>7788</td>
<td></td>
</tr>
</tbody>
</table>

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
Swizzling

<table>
<thead>
<tr>
<th>Memory</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>block 1</td>
<td>block 1</td>
</tr>
<tr>
<td>block 2</td>
<td>block 2</td>
</tr>
<tr>
<td>Rec A</td>
<td>Rec A</td>
</tr>
</tbody>
</table>

One Option:

<table>
<thead>
<tr>
<th>Translation Table</th>
<th>DB Addr</th>
<th>Mem Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rec-A</td>
<td>Rec-A-inMem</td>
<td></td>
</tr>
</tbody>
</table>

Another Option:

In memory pointers - need "type" bit

- to disk
- M to memory
Swizzling
• Automatic
• On-demand
• No swizzling / program control

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

Which is right for me?

Issues:
Flexibility Space Utilization
Complexity Performance
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
How to find a record quickly, given a key