CS 245: Database System Principles

Notes 03: Disk Organization

Hector Garcia-Molina

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

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

```
00000000 00100011
```

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

To represent:
- Characters
  → various coding schemes suggested, most popular is ascii

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

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

**String of characters**
- Null terminated
  - e.g., cat
- Length given
  - e.g., 3 cat
- Fixed length

**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

55 | Smith | 02
83 | Jones | 01

Records
Variable format

- Record itself contains format “Self Describing”

Example: variable format and length

<table>
<thead>
<tr>
<th># Fields</th>
<th>Code identifying field as E#</th>
<th>Integer Type</th>
<th>Code for Ename</th>
<th>String Type</th>
<th>Length of str.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5 1 46</td>
<td>4 5 2 6 F O R D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

3 E_name: Fred Child: Sally Child: Tom

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

John Sailing Chess --

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:

Example: Include record type in record

| 5 | 27 | ... |

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

Record header - data at beginning that describes record

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

Exercise: How to store XML data?

```xml
<table>
  <description> people on the fourth floor </description>
  <people>
    <person>
      <name> Alan </name>
      <age> 42 </age>
      <email> agb@abc.com </email>
    </person>
    <person>
      <name> Sally </name>
      <age> 30 </age>
      <email> sally@abc.com </email>
    </person>
  </people>
</table>
```

Other interesting issues:

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

Encrypting Records

trusted processor  \( E(r) \)  \( \text{dbms} \)

search \( F(r) = x \)  \( \text{trusted processor} \)  \( ?? \)  \( \text{dbms} \)
Search Key in the Clear

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

Encrypt Key

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

Issues

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

Solution?

- Develop new decryption function: \(\text{D}(f(k_1), E(k_2, \text{rand}))\) is true if \(k_1=k_2\)
Issues?

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

What are choices/issues with data compression?

• Leaving search keys uncompressed not as bad
• Larger compression units:
  - better for compression efficiency
  - worse for decompression overhead
• Similar data compresses better
  - compress columns?

Next: placing records into blocks

blocks ... a file

Options for storing records in blocks:

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

Next: placing records into blocks

blocks ... a file

(1) Separating records

Block

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

assume fixed length blocks

assume a single file (for now)
(2) Spanned vs. Unspanned

- Unspanned: records must be within one block
  block 1       block 2
  R1  R2  R3  R4  R5  R6  R7  ...

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

With spanned records:

\[ \begin{array}{cccccccc}
R1 & R2 & R3 & (a) & R3 & (b) & R4 & R5 & R6 & R7 & (a) \\
\end{array} \]

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

(3) 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

\[ \begin{array}{cccccccc}
R1 & Next (R1) & \ldots \\
\end{array} \]

(b) Linked

\[ \begin{array}{cccccccc}
R1 & Next (R1) \\
\end{array} \]
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?

\[ \rightarrow \text{Rx} \]

Many options:
Physical \quad \longleftrightarrow \quad \text{Indirect}

★ Purely Physical

E.g., Record Address or ID = \begin{cases} \text{Device ID} \\ \text{Cylinder #} \\ \text{Track #} \\ \text{Block #} \\ \text{Offset in block} \end{cases} \text{Block ID}

★ Fully Indirect

E.g., Record ID is arbitrary bit string

map

rec ID \[ r \]

Rec ID \text{Physical addr.} \quad \rightarrow \quad \text{address} \[ a \]
Tradeoff

Flexibility  ---  Cost

(For deletions, insertions)

Physical  ---  Indirect

Many options in between ...

Example: 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 ...

Options for storing records in blocks:

1. Separating records
2. Spanned vs. unspanned
3. Sequencing
4. Indirection

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
Options for Hosting

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

Tenants have similar data

customer | A | B | C | D | E | F
---|---|---|---|---|---|---
tenant 1: | a1 | b1 | c1 | d1 | e1 | -
          | a2 | b2 | c2 |   | e2 | f2

customer | A | B | C | D | G
---|---|---|---|---|---
tenant 2: | a3 | b3 | c2 |   |   | -
          | a1 | b1 | c1 | g1 |   | -
          | a4 |   |   |   | d1 |   

customer | A | B | C | D | E | F
---|---|---|---|---|---|---
tenant 1: | a1 | b1 | c1 | d1 | e1 | -
          | a2 | b2 | c2 |   | e2 | f2

customer | A | B | C | D | G
---|---|---|---|---|---
tenant 2: | a3 | b3 | c2 |   |   | -
          | a1 | b1 | c1 | g1 |   | -
          | a4 |   |   |   | d1 |   

salesforce.com solution

<table>
<thead>
<tr>
<th>customer</th>
<th>tenant</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a3</td>
<td>b3</td>
<td>c2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
</tbody>
</table>

Salesforce.com solution:

<table>
<thead>
<tr>
<th>customer</th>
<th>tenant</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td></td>
<td>e2</td>
<td>f2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer</th>
<th>tenant</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a3</td>
<td>b3</td>
<td>c2</td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>g1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>g1</td>
<td>d1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other Topics

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

Options:

(a) Immediately reclaim space
(b) Mark deleted

Deletion

Block

Rx
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
E.g., Leave “MARK” in map or old location

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

This space can be re-used
This space never 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...

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

Memory

Disk

block 1  
Rec A  
block 2

Swizzling

Memory

Disk

block 1  
Rec A  
block 2

Row vs Column Store

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

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  prod1  store1  price1  qty1
  id2  cust2  prod2  store2  price2  qty2
  id3  cust3  prod3  store3  price3  qty3
  id4  cust4  prod4  store4  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)
Interesting paper to read:


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