CSE 562
Database Systems

Disk Organization

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson Prentice Hall 2nd Edition
©2008 Garcia-Molina, Ullman, and Widom

cse@buffalo

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

\[
\begin{array}{ccc}
00000000 & 00100011 \\
\end{array}
\]

• 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 ...

To represent:

- **Dates**
  e.g.: - Integer, # days since Jan 1, 1900
  - 8 characters, YYYYMMDD
  - 7 characters, YYYYDDD

- **Time**
  e.g. - Integer, seconds since midnight
  - characters, HHMMSSFF

To represent:

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

- Bag of bits

| Length | 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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>smith</td>
<td>02</td>
</tr>
<tr>
<td>83</td>
<td>jones</td>
<td>01</td>
</tr>
</tbody>
</table>
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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td>F</td>
<td>O</td>
<td>R</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></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 | -- |

- 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 ...

Other interesting issues:

- Compression
  - within record - e.g. code selection
  - collection of records - e.g. find common patterns
- Encryption
Next: placing records into blocks

blocks ... a file

assume fixed length blocks
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
(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  [ ]  R3
  
  block 2
  R3  R4  R5  ...

- Spanned
  
  block 1
  R1  R2  R3 (a)  R3 (b)  R4  R5
  
  block 2
  R3 (b)  R4  R5  R6  R7 (a)  ...
With spanned records:

\[
\begin{array}{ccccccc}
\text{R1} & \text{R2} & \text{R3} & \text{R4} & \text{R5} & \text{R6} & \text{R7} \\
\text{(a)} & \text{(b)} & \text{(c)} & \text{(d)} & \text{(e)} & \text{(f)} & \text{(g)}
\end{array}
\]

- 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 (⇒ 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

R1 | Next (R1) |

(b) Linked

R1 | Next (R1) |

Sequencing Options

(c) Overflow area

Records in sequence

header

R1
R2
R3
R4
R5

(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 ID \}

Offset in block
**Fully Indirect**

E.g., Record ID is arbitrary bit string

```
<table>
<thead>
<tr>
<th>Rec ID</th>
<th>Physical addr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>a</td>
</tr>
</tbody>
</table>
```

**Tradeoff**

Flexibility $\leftrightarrow$ Cost of indirection (for deletions, insertions)

**Example: Indirection in block**

A block:

- Header
- Free space
- R3
- R4
- R1
- R2

Physical $\leftrightarrow$ Indirect

Many options in between ...
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

Options for Hosting
- Separate DBMS per tenant
- One DBMS, separate tables per tenant
- One DBMS, shared tables
Tenants Have Similar Data

<table>
<thead>
<tr>
<th>customer</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>tenant 1:</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td></td>
<td>-</td>
<td>e2</td>
</tr>
<tr>
<td>tenant 2:</td>
<td>a3</td>
<td>b2</td>
<td>c3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
<td>g1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a4</td>
<td></td>
<td></td>
<td></td>
<td>d1</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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>a3</td>
<td>b3</td>
<td>c2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cust-other</th>
<th>tenant</th>
<th>A</th>
<th>f1</th>
<th>v1</th>
<th>f2</th>
<th>v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>D</td>
<td>d1</td>
<td>E</td>
<td>e1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a2</td>
<td>E</td>
<td>e2</td>
<td>F</td>
<td>f2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>G</td>
<td>g1</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a4</td>
<td>D</td>
<td>g1</td>
<td>d1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

```
R1 -> ?
```
Solution #2: Tombstones

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

- **Physical IDs**

  A block

  ![Diagram](Diagram1.png)

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

- **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...

**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!

Swizzling
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)

Column Store

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

<table>
<thead>
<tr>
<th>id1</th>
<th>cust1</th>
<th>prod1</th>
<th>store1</th>
<th>price1</th>
<th>date1</th>
<th>qty1</th>
</tr>
</thead>
<tbody>
<tr>
<td>id2</td>
<td>cust2</td>
<td>prod2</td>
<td>store2</td>
<td>price2</td>
<td>date2</td>
<td>qty2</td>
</tr>
<tr>
<td>id3</td>
<td>cust3</td>
<td>prod3</td>
<td>store3</td>
<td>price3</td>
<td>date3</td>
<td>qty3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(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:

- http://www.cs.umb.edu/%7Eponeil/vidb05_cstore.pdf

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

This Time

- Hardware
  - Chapter 13: 13.5-13.8