Data Storage Formats

Instructor: Matei Zaharia

cs245.stanford.edu
Outline

Overview

Record encoding

Collection storage

Indexes
Outline

Overview

Record encoding

Collection storage

Indexes
Overview

Recall from last time: I/O slow compared to compute, random I/O ≪ sequential

Key concerns in storage:

» **Access time**: minimize # of random accesses, bytes transferred, etc
  • Main way: place co-accessed data together!

» **Size**: storage costs $

» **Ease of updates**
General Setup

Record collection

Index

Secondary index

...
Outline

Overview

Record encoding

Collection storage

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

Real, floating point

n bits for mantissa, m for exponent....
To Represent:

Characters

→ Various coding schemes available

Example: ASCII
A: 1000001
a: 1100001
5: 0110101
LF: 0001010
To Represent:

Boolean

e.g., TRUE  \[
\begin{array}{c}
1111 \\
1111
\end{array}
\]
FALSE  \[
\begin{array}{c}
0000 \\
0000
\end{array}
\]

Application specific

e.g., RED $\rightarrow$ 1  GREEN $\rightarrow$ 3
BLUE $\rightarrow$ 2  YELLOW $\rightarrow$ 4  ...

\[\text{\Rightarrow Can we use less than 1 byte/code?}\]
Yes, but only if desperate...
To Represent:

Dates

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

Time

e.g. - Integer, seconds since midnight
        - characters, HHMMSSFF
To Represent:

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

| 3 | c | a | t | X |
To Represent:

<table>
<thead>
<tr>
<th>Bag of bits</th>
<th>Length</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To Represent:

N O T H I N G

A B S O L U T E L Y N O T H I N G

T H I S I S A M E M E

A B O U T N O T H I N G
To Represent: Nothing

NULL concept in SQL (not same as 0 or “”) 

Physical representation options:
» Special “sentinel” value in fixed-length field 
» Boolean “is null” flag
» Just skip the field in a sparse record format

Pretty common in practice!
Key Point

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

Type of an item: tells us how to interpret the bytes, plus size if fixed
Bigger Collections

Data Items

Records

Blocks

Files
Record: Set of Related Data Items ("Fields")

E.g.: Employee record:

- name field,
- salary field,
- date-of-hire field, ...

CS 245
Types of Records

Main choices:
  » Fixed vs variable **format**
  » Fixed vs variable **length**
Fixed Format

A schema (not record) contains following info:

- # of fields
- type of each field
- order in record
- meaning of each field
Example: Fixed Format & Length

Employee record

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

\[
\begin{array}{c|c|c}
55 & smith & 02 \\
83 & jones & 01 \\
\end{array}
\]
Variable Format

Record itself contains format

“Self Describing”
Example: Variable Format & 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 Repeated Fields**

Employee $\rightarrow$ one or more $\rightarrow$ children

| 3 | E_name: Fred | Child: Sally | Child: Tom |
Note: Repeated Fields Does Not Imply Variable Format/Length

Could have fixed space for a max # of items and their sizes

| John | Sailing | Chess | (null) |
Many Variants Between Fixed and Variable Format

Example: Include a record type in record

| 5 | 27 | . . . |

Type is a pointer to one of several schemas
Record Header: Data at Start that Describes a Record

May contain:

- record type
- record length
- timestamp
- concurrency stuff ...
Exercise: How to store XML data?

<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>

Source: Data on the Web, Abiteboul et al
Other Interesting Issues

Compression
  » Within record: e.g. encoding selection
  » Collection of records: use common patterns

Encryption
  » Usually operates on large blocks
Outline

Overview

Record encoding

Collection storage

Indexes
Collection Storage Questions

How do we place data items and records for efficient access?
  » **Locality** and **searchability**

How do we physically encode records in blocks and files?
Placing Data for Efficient Access

**Locality:** which items are accessed together
  » When you read one field of a record, you’re likely to read other fields of the same record
  » When you read one field of record 1, you’re likely to read the same field of record 2

**Searchability:** quickly find relevant records
  » E.g. sorting the file lets you do binary search
## Locality Example: Row Stores vs Column Stores

### Row Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Fields stored contiguously in one file

### Column Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Each column in a different file
Locality Example: Row Stores vs Column Stores

Row Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Fields stored contiguously in one file

Column Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Each column in a different file

Accessing all fields of one record: 1 random I/O for row, 3 for column
Locality Example: Row Stores vs Column Stores

### Row Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Fields stored contiguously in one file.

### Column Store

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

Each column in a different file.

Accessing one field of all records: 3x less I/O for column store.
Can We Have Hybrids Between Row & Column?

Yes! For example, colocated *column groups*:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>20</td>
<td>CA</td>
</tr>
<tr>
<td>Bob</td>
<td>30</td>
<td>CA</td>
</tr>
<tr>
<td>Carol</td>
<td>42</td>
<td>NY</td>
</tr>
<tr>
<td>David</td>
<td>21</td>
<td>MA</td>
</tr>
<tr>
<td>Eve</td>
<td>26</td>
<td>CA</td>
</tr>
<tr>
<td>Frances</td>
<td>56</td>
<td>NY</td>
</tr>
<tr>
<td>Gia</td>
<td>19</td>
<td>MA</td>
</tr>
<tr>
<td>Harold</td>
<td>28</td>
<td>AK</td>
</tr>
<tr>
<td>Ivan</td>
<td>41</td>
<td>CA</td>
</tr>
</tbody>
</table>

File 1 File 2: age & state

Helpful if age & state are frequently co-accessed
Improving Searchability: Ordering

Ordering the data by a field will give:

» Closer I/Os if queries tend to read data with nearby values of the field (e.g. time ranges)
» Option to accelerate search via an ordered index (e.g. B-tree), binary search, etc

What’s the downside of having an ordering?
Improving Searchability: Partitions

Just place data into buckets based on a field (but not necessarily fine-grained order)

E.g. Hive table storage over filesystem or S3:

/my_table/date=20190101/file1.parquet
/my_table/date=20190101/file2.parquet
/my_table/date=20190102/file1.parquet
/my_table/date=20190102/file2.parquet
/my_table/date=20190103/file1.parquet
...

Easy to add, remove & list files in any directory
Can We Have Searchability on Multiple Fields at Once?

Yes! Many possible ways:

1) Multiple partition or sort keys (e.g. partition data by date, then group by customer ID)

2) Interleaved orderings such as Z-ordering
Z-Ordering

How Do We Encode Records into Blocks & Files?
How Do We Encode Records into Blocks & Files?

blocks

records

a file
Questions in Storing Records

(1) separating records
(2) spanned vs. unspanned
(3) 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:

<table>
<thead>
<tr>
<th>block 1</th>
<th>block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>R3</td>
</tr>
<tr>
<td>R2</td>
<td>R4</td>
</tr>
<tr>
<td>R3 (a)</td>
<td>R5</td>
</tr>
<tr>
<td>R3 (b)</td>
<td>R6</td>
</tr>
<tr>
<td>R4</td>
<td>R7 (a)</td>
</tr>
<tr>
<td>R5</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
</tr>
<tr>
<td>R7 (a)</td>
<td></td>
</tr>
</tbody>
</table>
With Spanned Records

![Diagram showing 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 storage space…

Spanned essential if record size > block size
(4) Indirection

How does one refer to specific records? (e.g. in metadata or in other records)

Rx
(4) Indirection

How does one refer to records?

Many options:

Physical \quad \leftrightarrow \quad \text{Indirect}

Rx
Purely Physical

E.g., Record Address = Device ID

or ID

Cylinder #

Track #

Block #

Offset in block

Block ID
Fully Indirect

E.g., Record ID is arbitrary bit string

![Diagram of Fully Indirect mapping]

- rec ID
- map
- address
- Rec ID
- Physical addr.
Tradeoff

Flexibility ↔ Cost

to move records of indirection
(for deletions, insertions)
Physical ↔ Indirect

Many options in between …
Example: Indirection in Block

A block:

Header

Free space

R3

R4

R1

R2
Block Header: Data at Start that Describes Block

May contain:
- File ID (or table or database ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4)
- Pointer to other blocks “like it”
- Timestamp ...
Other Concern: Deletion!
Options

(a) Immediately reclaim space

(b) Mark deleted
Options

(a) Immediately reclaim space

(b) Mark deleted
   - May need chain of deleted records
     (for space re-use)
   - Need a way to mark:
     • special characters
     • delete field
     • entries in maps
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 1: Do Not Worry
Solution 2: Tombstones

Special mark in old location or mappings
Solution 2: Tombstones

Special mark in old location or mappings

Physical IDs:

A block

This space can never re-used

This space can be re-used
Solution 2: Tombstones

Special mark in old location or mappings

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...
Insertion

**Easy case:** records not ordered

→ Insert new record at end of file or in a deleted slot

→ If records are variable size, not as easy...
Insertion

Hard case: records are ordered

→ If free space close by, not too bad...

→ Otherwise, use an overflow area?
Interesting Problems

How much free space to leave in each block, track, cylinder?

How often do I reorganize file + overflow?
Summary

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 Strategy, Compute:

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