Data Storage Formats

Instructor: Matei Zaharia
Outline

Overview

Record encoding

Collection storage

C-Store paper

Indexes
Outline

Overview

Record encoding

Collection storage

C-Store paper

Indexes
Designing Storage Formats

Key concerns:

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

» **Space:** storage costs $

» **Ease of updates**
General Setup

Record collection

Index

Secondary index

...
Outline

Storage devices wrap-up

Record encoding

Collection storage

C-Store paper

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

What we have available: 8 bits
Fixed-Length Items

Integer: fixed # of bytes (e.g., 2 bytes)

  e.g., 35 is 00000000 00100011

Floating-point: n-bit mantissa, m-bit exponent

Character: encode as integer (e.g. ASCII)
Variable-Length Items

String of characters:
  » Null-terminated  
    | c a t |
  » Length + data  
    | 3 c a t |
  » Fixed-length

Bag of bits:  

| Length | Bits |
Representing
Representing 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!
Bigger Collections

Data Items

Records

Blocks

Files
Record: Set Data Items (Fields)

E.g. employee record:
» name field
» salary field
» date-of-hire field
» ...
Record Encodings

Fixed vs variable format

Fixed vs variable length
Fixed Format

A schema for all records in table specifies:

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

Employee record

(1) EID, 2 byte integer

(2) Name, 10 chars

(3) Dept, 2 byte code

55 smith 02

83 jones 01

Schema

Records
Variable Format

Record itself contains format

“Self-describing”
Example: Variable Format & Length

# Fields
Code identifying field as EID
Integer type

Code for name
String type
Length of str.

<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>I</th>
<th>46</th>
<th>4</th>
<th>S</th>
<th>4</th>
<th>F</th>
<th>O</th>
<th>R</th>
<th>D</th>
</tr>
</thead>
</table>

Variable Format Useful For

“Sparse” records
Repeating fields
Evolving formats

But may waste space...
Many Variants Between Fixed and Variable Format

Example: Include a record type in record

| 5 | 27 | . . . |

record type  record length

Type is a pointer to one of several schemas
Outline

Overview

Record encoding

Collection storage

C-Store paper

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

/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 by date, then sort by userID)

2) Interleaved orderings such as Z-ordering
Z-Ordering
How Do We Encode Records into Blocks & Files?
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:

need indication of partial record
(3) Indirection

How does one refer to other records?

Many options:

Physical  Indirect
Purely Physical

E.g., Record Address = [Device ID, Cylinder #, Track #, Block #, Offset in block]

or ID
Fully Indirect

E.g., Record ID is arbitrary bit string
Tradeoff

Flexibility \quad \leftrightarrow \quad \text{Cost}

to move records \quad \text{of indirection}

(for deletions, insertions)
Inserting Records

**Easy case:** records not ordered
  » Insert record at end of file or in a free space
  » Harder if records are variable-length

**Hard case:** records are ordered
  » If free space close by, not too bad...
  » Otherwise, use an **overflow** area and reorganize the file periodically
Deleting Records

Immediately reclaim space

OR

Mark deleted
  – And keep track of freed spaces for later use
Interesting Problems

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

How often to reorganize file + merge overflow?
Compressing Collections

Usually for a block at a time
  » Benefits from placing similar items together

Can be integrated with execution (C-Store)
Summary

There are many ways to organize data on disk

Key tradeoffs:

Flexibility  ________  Space Utilization

Complexity  ________  Performance
To Evaluate a Strategy, Compute:

Space used for expected data

Expected time to

- fetch record given key
- read whole file
- insert record
- delete record
- update record
- reorganize file
- ...
“Integrating Compression and Execution in Column-Oriented Database Systems”

From the MIT C-Store project (led to Vertica)