NoDB / RAW

Oct 8, 2017
Why?

• Control over data.
  • The database needs the canonical version
• Load times
  • Copying data is expensive.
External Tables

- Supported by SQLite, Postgres, Oracle, DB2, ...
- Read only access to raw data files
  - ... but very slow
NoDB / RAW

- Parsing is slow
  - ... so cut out unnecessary parsing steps.
  - ... so cache intermediate parsing metadata.
  - ... so just-in-time compile extraction code.
- Raw formats not optimized for database access.
  - ... so cache parsed data in the database.
Data Organization

- How do we store data?
  - How are records represented on-disk? (Serialization)
  - How are records stored within a page?
  - How are pages organized in a file?
  - What other metadata do we need?
- Our solutions must also be persisted to disk.
Record (Tuple) Formats

Base Address (B)  \[ \text{Address } B + L1 + L2 \]
Record (Tuple) Formats

Number of Fields

Delimiters
Record (Tuple) Formats

Array of Field Offsets
Page Formats

Packed

Data Records

Free Space

Unpacked, Bitmap

Bit array of occupied slots (and size of page)

Number of records

01101011...1 N
Page Formats

Variable Size Records

Pointer to start of free space
## Anatomy of a CSV file

<table>
<thead>
<tr>
<th>Year</th>
<th>Make</th>
<th>Model</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>Ford</td>
<td>E350</td>
<td>ac, abs, moon</td>
<td>3000.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;Extended Edition&quot;</td>
<td></td>
<td>4900.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;Extended Edition, Very Large&quot;</td>
<td></td>
<td>5000.00</td>
</tr>
<tr>
<td>1996</td>
<td>Jeep</td>
<td>Grand Cherokee</td>
<td>MUST SELL! air, moon roof, loaded</td>
<td>4799.00</td>
</tr>
</tbody>
</table>

Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
### The Filesystem View

<table>
<thead>
<tr>
<th>Year</th>
<th>Make</th>
<th>Model</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>Ford</td>
<td>E350</td>
<td>ac, abs, moon</td>
<td>3000.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;&quot;Extended Edition&quot;&quot;, &quot;&quot;</td>
<td>4900.00</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;&quot;Extended Edition, Very Large&quot;&quot;,</td>
<td>5000.00</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>Jeep</td>
<td>Grand Cherokee</td>
<td>MUST SELL! air, moon roof, loaded</td>
<td>4799.00</td>
</tr>
</tbody>
</table>
The Filesystem View

Year, Make, Model, Description, Price

1997, Ford, E350, "ac, abs, moon", 3000.00

1999, Chevy, "Venture ""Extended Edition""", 4900.00

1999, Chevy, "Venture ""Extended Edition, Very Large"", 5000.00

1996, Jeep, Grand Cherokee, "MUST SELL! air, moon roof, loaded", 4799.00
## The Filesystem View

<table>
<thead>
<tr>
<th>Year</th>
<th>Make</th>
<th>Model</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>Ford</td>
<td>E350</td>
<td>&quot;ac, abs, moon&quot;</td>
<td>3000.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;&quot;Extended Edition&quot;&quot;&quot;&quot;</td>
<td>4900.00</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;&quot;Extended Edition, Very Large&quot;&quot;&quot;&quot;</td>
<td>5000.00</td>
<td></td>
</tr>
<tr>
<td>1996</td>
<td>Jeep</td>
<td>Grand Cherokee</td>
<td>&quot;MUST SELL!&quot;air, moon roof, loaded&quot;</td>
<td>4799.00</td>
</tr>
</tbody>
</table>
Field Parsing

https://sourcemaking.com/design_patterns/state/delphi
Field Parsing

1997, Ford, E350, "ac, abs, moon", 3000.00

1999, Chevy, "Venture ""Extended Edition"", "", 4900.00

1999, Chevy, "Venture ""Extended Edition, Very Large"", "", 5000.00

1996, Jeep, Grand Cherokee, "MUST SELL!\nair, moon roof, loaded", 4799.00
Field Parsing

1997, Ford, E350, "ac, abs, moon", 3000.00
1999, Chevy, "Venture ""Extended Edition"", "", 4900.00
1999, Chevy, "Venture ""Extended Edition, Very Large"", 5000.00
1996, Jeep, Grand Cherokee, "MUST SELL!\nair, moon roof, loaded", 4799.00
Field Parsing

1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""",",4900.00
1996,Jeep,Grand Cherokee,"MUST SELL!\nair, moon roof, loaded",4799.00
Overview

Rows → Fields → Typed Data
Overview

Core Idea: Do this part once only
Avoiding Re-Splitting

L1  L2  L3  L4  L5

R1

R2

R3

1 2 3 4 ...
Avoiding Re-Splitting
Position Map

**Idea 1:** Index the data
Year, Make, Model, Description, Price
1997, Ford, E350, "ac, abs, moon", 3000.00
1999, Chevy, "Venture "Extended Edition"", 4900.00
1999, Chevy, "Venture "Extended Edition, Very Large"", 5000.00
1996, Jeep, Grand Cherokee, "MUST SELL!" air, moon roof, loaded", 4799.00
Position Map

Year, Make, Model, Description, Price

1997, Ford, E350, "ac, abs, moon", 3000.00
1999, Chevy, "Venture "Extended Edition"
1999, Chevy, "Venture "Extended Edition, Very Large"
1996, Jeep, Grand Cherokee, "MUST SELL!" air, moon roof, loaded", 4799.00
Overview

Rows → Fields → Typed Data
Overview

Rows

Fields

Typed Data

Typed Data

Typed Data

Typed Data

Typed Data

Typed Data

Typed Data

Typed Data
Overview
Position Maps

Idea 2: Index Lazily
<table>
<thead>
<tr>
<th>Year</th>
<th>Make</th>
<th>Model</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>Ford</td>
<td>E350</td>
<td>ac, abs, moon</td>
<td>3000.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;Extended Edition&quot;&quot;&quot;</td>
<td></td>
<td>4900.00</td>
</tr>
<tr>
<td>1999</td>
<td>Chevy</td>
<td>&quot;Venture &quot;Extended Edition, Very Large&quot;&quot;&quot;</td>
<td></td>
<td>5000.00</td>
</tr>
<tr>
<td>1996</td>
<td>Jeep</td>
<td>Grand Cherokee</td>
<td>MUST SELL!\nair, moon roof, loaded</td>
<td>4799.00</td>
</tr>
</tbody>
</table>
SELECT Make FROM Cars
SELECT Make FROM Cars
SELECT Year FROM Cars WHERE Make = 'Chevy'

Year, Make, Model, Description, Price
1997, Ford, E350, "ac, abs, moon", 3000.00
1999, Chevy, "Venture ""Extended Edition""", 4900.00
1999, Chevy, "Venture ""Very Large"", 5000.00
1996, Jeep, Grand Cherokee, "MUST SELL!\nair, moon roof, loaded", 4799.00
Position Maps

**Idea 3**: Cache “small” parsed values
Position Map

<table>
<thead>
<tr>
<th>Record 1</th>
<th>Record 2</th>
<th>Record 3</th>
<th>Record 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1997,3000.00&gt;</td>
<td>&lt;1999,4900.00&gt;</td>
<td>&lt;1999,5000.00&gt;</td>
<td>&lt;1996,4799.00&gt;</td>
</tr>
</tbody>
</table>

Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon", 3000.00
1999,Chevy,"Venture "Extended Edition"",,4900.00
1996,Jeep,Grand Cherokee,"MUST SELL!"nair, moon roof, loaded",4799.00
Performance
JIT-ed Parsers

Idea 4: Eliminate Branching
Branch Prediction

for every column {
    char *raw // raw data
    Datum *datum // loaded data

    // read field from file
    raw = readNextFieldFromFile(file)

    switch (schemaDataType[column]) {
        case IntType: datum = convertToInteger(raw)
        case FloatType: datum = convertToFloat(raw)
        ...
    }
}
for every column {
    char *raw // raw data
    Datum *datum // loaded data

    // read field from file
    raw = readNextFieldFromFile(file)

    switch (schemaDataType[column]) {
    case IntType: datum = convertToInteger(raw)
    case FloatType: datum = convertToFloat(raw)
    ...
    }
}
JIT-ed Parsers

for every column {
    char *raw // raw data
    Datum *datum // loaded data

    // read field from file
    raw = readNextFieldFromFile(file)

    switch (schemaDataType[column]) {
        case IntType: datum = convertToInteger(raw)
        case FloatType: datum = convertToFloat(raw)
        ...
    }
}
for every column {
  char *raw // raw data
  Datum *datum // loaded data

  // read field from file
  raw = readNextFieldFromFile(file)

  switch (schemaDataType[column]) {
    case IntType: datum = convertToInteger(raw)
    case FloatType: datum = convertToFloat(raw)
    ...
JIT-ed Parsers

Datum *datum[] // loaded data

datum[0] = convertToInteger(readNextFieldFromFile(file));
datum[1] = convertToString(readNextFieldFromFile(file));
datum[2] = convertToString(readNextFieldFromFile(file));
datum[3] = convertToString(readNextFieldFromFile(file));
datum[4] = convertToFloat(readNextFieldFromFile(file));

No Loop, No Ifs, No branching
More opportunities

Base Address (B) Address B + L1 + L2
JIT-ed Parsers

char *raw // raw data for record
Datum *datum[] // loaded data

datum[0] = convertToInteger(raw[0]);
datum[1] = convertToFloat(raw[4]);
datum[2] = convertToShort(raw[8]);
datum[4] = convertToFloat(raw[10]);