Columnar Databases
What, why and how

Christopher Strecker

October 2014
Roadmap

Motivation

Implementation I: Data Storage
  Columnar Storage
  Compression

Implementation II: Query Execution
  Lazy Decompression
  Batch Processing
  Late Materialization
  Online Indexing

Summary
Benchmark
Real world data

[Benchmark graph showing how queries run faster with MonetDB than with PostgreSQL]
Column Stores?

- **Same interface** as traditional relational databases:
  - Structured data
  - Structured Query Language
  - Atomicity, Consistency, Integrity, Durability

- **Optimized for reading** large amounts of data:
  - Good OLAP performance
  - Not competitive on OLTP performance

- Differences in data storage and query execution
**Row store ↔ Column store**

<table>
<thead>
<tr>
<th>OrderID</th>
<th>UserID</th>
<th>Product</th>
<th>Revenue</th>
<th>Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1005</td>
<td>1</td>
<td>103.25</td>
<td>5.00</td>
</tr>
<tr>
<td>10020</td>
<td>34295</td>
<td>1</td>
<td>105.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10033</td>
<td>124</td>
<td>5</td>
<td>92.70</td>
<td>5.00</td>
</tr>
<tr>
<td>10051</td>
<td>8399</td>
<td>5</td>
<td>100.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

...
Use Domain Knowledge

- Column data often has less “randomness”
  ⇒ Better compression can be found
  ⇒ Even less I/O
Compressing Columns

- Dictionary encoding for Voucher: {0: 0.00, 1: 5.00}
Compressing Columns

- Dictionary encoding for Voucher: \{0: 0.00, 1: 5.00\}
- Delta encoding for Revenue: revenue = 100.00 + x

<table>
<thead>
<tr>
<th>OrderID</th>
<th>UserID</th>
<th>Product</th>
<th>Revenue</th>
<th>Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1005</td>
<td>1</td>
<td>3.25</td>
<td>1</td>
</tr>
<tr>
<td>10020</td>
<td>34295</td>
<td>1</td>
<td>5.00</td>
<td>0</td>
</tr>
<tr>
<td>10033</td>
<td>124</td>
<td>5</td>
<td>-7.30</td>
<td>1</td>
</tr>
<tr>
<td>10051</td>
<td>8399</td>
<td>5</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Compressing Columns

- Dictionary encoding for **Voucher**: {0: 0.00, 1: 5.00}
- Delta encoding for **Revenue**: revenue = 100.00 + x
- Run length encoding for **Product**
## Compressing Columns

- Dictionary encoding for Voucher: \{0: 0.00, 1: 5.00\}
- Delta encoding for Revenue: revenue = 100.00 + x
- Run length encoding for Product
- Compression for UserID?

<table>
<thead>
<tr>
<th>OrderID</th>
<th>UserID</th>
<th>Product</th>
<th>Revenue</th>
<th>Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1005</td>
<td>(2, 1)</td>
<td>3.25</td>
<td>1</td>
</tr>
<tr>
<td>10020</td>
<td>34295</td>
<td>(2, 5)</td>
<td>5.00</td>
<td>0</td>
</tr>
<tr>
<td>10033</td>
<td>124</td>
<td></td>
<td>-7.30</td>
<td>1</td>
</tr>
<tr>
<td>10051</td>
<td>8399</td>
<td></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Compressing Columns

- Dictionary encoding for Voucher: \{0: 0.00, 1: 5.00\}
- Delta encoding for Revenue: revenue = 100.00 + x
- Run length encoding for Product
- Compression for UserID?
- Delta encoding for OrderID: OrderID = PreviousID + x

<table>
<thead>
<tr>
<th>OrderID</th>
<th>UserID</th>
<th>Product</th>
<th>Revenue</th>
<th>Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1005</td>
<td>(2, 1)</td>
<td>3.25</td>
<td>1</td>
</tr>
<tr>
<td>+19</td>
<td>34295</td>
<td>(2, 5)</td>
<td>5.00</td>
<td>0</td>
</tr>
<tr>
<td>+13</td>
<td>124</td>
<td></td>
<td>-7.30</td>
<td>1</td>
</tr>
<tr>
<td>+18</td>
<td>8399</td>
<td></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Storage Level Changes

Summary

- I/O is often the bottleneck in OLAP queries
- Software can **not** make I/O faster
- Therefore, reduce I/O:
  - Only read necessary columns
  - Use knowledge of data structure for better compression
Motivation

Implementation I: Data Storage
  Columnar Storage
  Compression

Implementation II: Query Execution
  Lazy Decompression
  Batch Processing
  Late Materialization
  Online Indexing

Summary
Lazy Decompression

Operating on Compressed Data

- Orders with voucher: **dictionary lookup, then select**
- Revenue: \((3.25 + 5.00 - 7.30 + 0) + (4 \cdot 100)\)
- # Products: read **run length**
- **Order ID, User ID?**

<table>
<thead>
<tr>
<th>OrderID</th>
<th>UserID</th>
<th>Product</th>
<th>Revenue</th>
<th>Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1005</td>
<td>(2, 1)</td>
<td>3.25</td>
<td>1</td>
</tr>
<tr>
<td>+19</td>
<td>34295</td>
<td>(2, 5)</td>
<td>5.00</td>
<td>0</td>
</tr>
<tr>
<td>+13</td>
<td>124</td>
<td></td>
<td>-7.30</td>
<td>1</td>
</tr>
<tr>
<td>+18</td>
<td>8399</td>
<td></td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**Single Instruction, Multiple Data**

```sql
SELECT sum(revenue) FROM sales_fact
JOIN product ON product_fk = product_id
WHERE product_name IN (1, 5);
```
Late Materialization

Keep Columns

- Present data to the user as rows
- Keep data internally in columns as long as possible

<table>
<thead>
<tr>
<th>Product</th>
<th>Bitmap</th>
<th>Revenue</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>104.66</td>
<td>495.84</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>109.46</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>98.30</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>91.78</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>98.03</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>98.56</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>91.38</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>92.39</td>
<td></td>
</tr>
</tbody>
</table>
Database Cracking

SELECT * FROM customer WHERE age > 50

Age
34
53
66
75
23
25
42
62
66
43

Age_
34
23
25
42
43
53
66
75
62
66

Side effect

Table-scan

≤ 50

> 50
Summary

- Keep Tables, SQL and ACID
- Significant speedup without distributed computation
- Transparent scaling to multiple machines
- Tradeoff: Optimizing for OLAP workload
References


