Columnar Databases

What, why and how

Christopher Strecker

October 2014

Roadmap

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

OrderID	UserID	Product	Revenue	Voucher
10001	1005	1	103.25	5.00
10020	34295	1	105.00	0.00
10033	124	5	92.70	5.00
10051	8399	5	100.00	0.00
...	...	...	...	...

OrderID
10001
10020
10033
10051
...

UserID
1005
34295
124
8399
...

Product
1
1
5
5
...

Revenue
103.25
105.00
92.70
100.00
...

Voucher
5.00
0.00
5.00
0.00
...

Use Domain Knowledge

- Column data often has less “randomness”

  ⇒ Better compression can be found

  ⇒ Even less I/O

OrderID
10001
10020
10033
10051
...

UserID
1005
34295
124
8399
...

Product
1
1
5
5
...

Revenue
103.25
105.00
92.70
100.00
...

Voucher
5.00
0.00
5.00
0.00
...

Compressing Columns

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

OrderID	UserID	Product	Revenue	Voucher
10001	1005	1	103.25	1
10020	34295	1	105.00	0
10033	124	5	92.70	1
10051	8399	5	100.00	0
...	...	...	...	...

Compressing Columns

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

OrderID	UserID	Product	Revenue	Voucher
10001	1005	1	3.25	1
10020	34295	1	5.00	0
10033	124	5	−7.30	1
10051	8399	5	0.00	0
...	...	...	...	...

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

OrderID	UserID	Product	Revenue	Voucher
10001	1005	(2, 1)	3.25	1
10020	34295		5.00	0
10033	124	(2, 5)	-7.30	1
10051	8399		0.00	0
...	...	...	...	...

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?

OrderID	UserID	Product	Revenue	Voucher
10001	1005	(2, 1)	3.25	1
10020	34295		5.00	0
10033	124	(2, 5)	−7.30	1
10051	8399		0.00	0
...	...	...	...	...

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

OrderID	UserID	Product	Revenue	Voucher
10001	1005	(2, 1)	3.25	1
+19	34295		5.00	0
+13	124	(2, 5)	−7.30	1
+18	8399		0.00	0
...	...	...	...	...

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

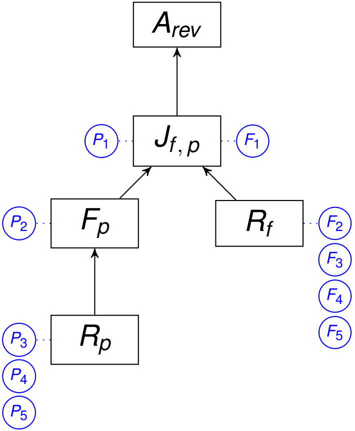
- Orders with voucher: dictionary lookup, then select
- Revenue: $(3.25 + 5.00 - 7.30 + 0) + (4 \cdot 100)$
- # Products: read run length
- Order ID, User ID?

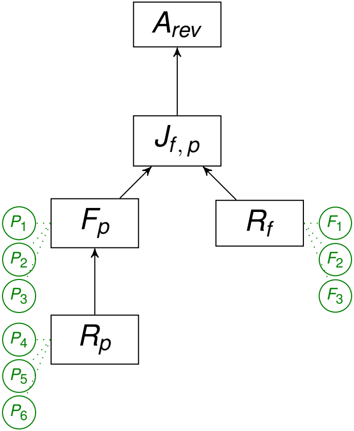
OrderID	UserID	Product	Revenue	Voucher
10001	1005	(2, 1)	3.25	1
+19	34295		5.00	0
+13	124	(2, 5)	−7.30	1
+18	8399		0.00	0
...	...	...	...	...

Batch Processing

Single Instruction, Multiple Data

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





Keep Columns

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

Product
1
5
3
5
2
1
1
4

→

Bitmap
1
1
0
1
0
1
1
0

→

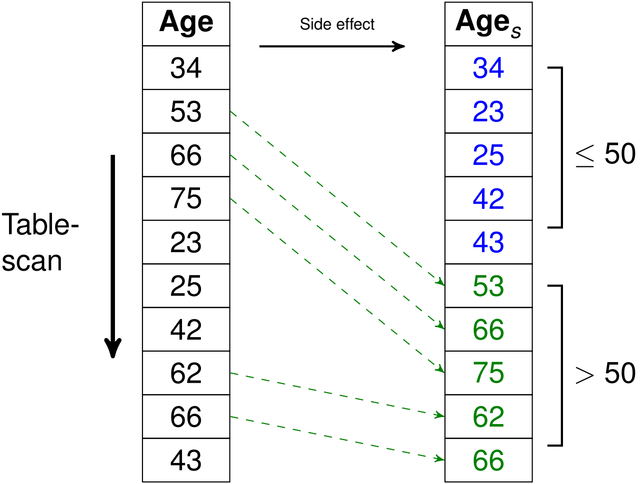
Revenue
104.66
109.46
98.30
91.78
98.03
98.56
91.38
92.39

→

Sum
495.84

Database Cracking

```
SELECT * FROM customer WHERE age > 50
```



Summary

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

References

[1] Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, and Samuel Madden.
The design and implementation of modern column-oriented database systems.
Technical report, 2013.

[2] Stratos Idreos, Martin L Kersten, and Stefan Manegold.
Database Cracking.
In *CIDR*, volume 7, pages 7–10, 2007.

[3] Software Engineering Radio.
Episode 199: Michael Stonebraker on Current Developments in Databases.

[4] Mike Stonebraker, Daniel J Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik.
C-store: a column-oriented DBMS.
In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.