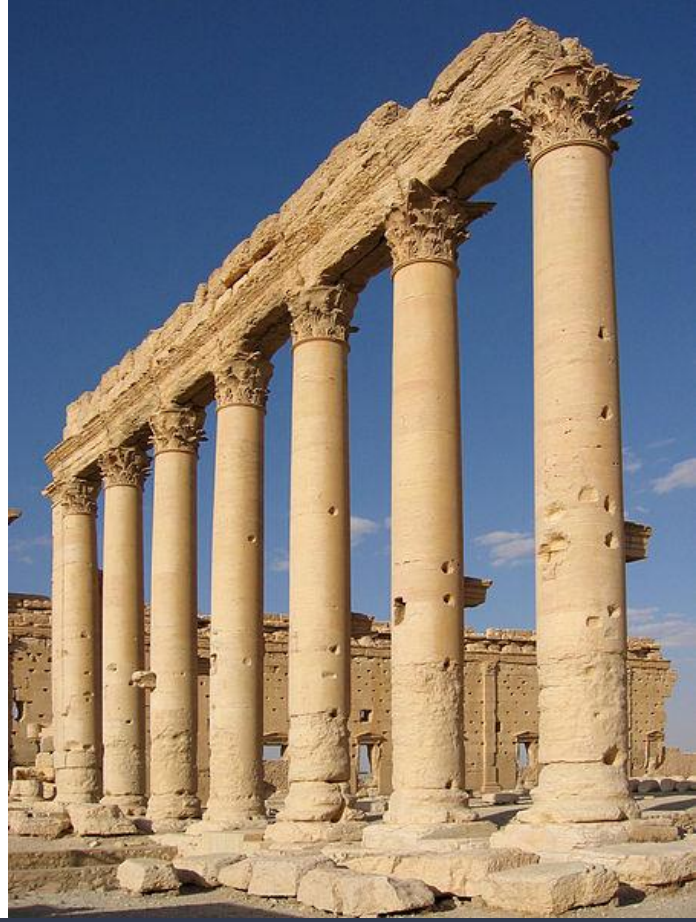
Columnar Databases

More than you probably wanted to know

Jared Briskman
and
Margo Crawford

VS



Row-Store vs Column-Store

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr.	Troy	VT
0003647	Flotsam	5 Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

VS

Record #	Name	Address	City	State
0003623	ABC	125 N Way	Cityville	PA
0003626	Newburg	1300 Forest Dr	Troy	VT
0003647	Flotsam	Industrial Pkwy	Springfield	MT
0003705	Jolly	529 S 5th St.	Anywhere	NY

Row-Store vs Column-Store

Column Orientation...

Affects:

- Physical-layer implementation
  - Performance tradeoffs
    - Messier to write to
    - Different query tuning
  - Better compression ability

Does Not Affect:

- Conceptual paradigm
- Inherent Quality
- Physical optimization tricks (indicies, caches, B+ trees, etc...)

Let’s take a look at a very simplified case.

- Ignoring most implementation details
- The same trends exist with optimizations
- Trends are illustrative, not accurate.

A simple example relation

ID	Name	Checking	Type	Savings
1	Alice	500	4	800
2	Bob	1000	5	300
3	Chris	300	1	400
5	David	10,000	1	20,000
6	Elaine	800	2	40
7	Frank	300	3	200
8	Greg	200	3	600

Where are the blocks - Row (Abstract)

ID	Name	Checking	Type	Savings
1	Alice	500	4	800
2	Bob	1000	5	300
3	Chris	300	1	400
5	David	10,000	1	20,000
6	Elaine	800	2	40
7	Frank	300	3	200
8	Greg	200	3	600

Where are the blocks - Columnar (Abstract)

ID	Name	Checking	Type	Savings
1	Alice	500	4	800
2	Bob	1000	5	300
3	Chris	300	1	400
5	David	10,000	1	20,000
6	Elaine	800	2	40
7	Frank	300	3	200
8	Greg	200	3	600

Where are the blocks - Row (4KB blocks)

ID : Int	Name : Str(8)	Checking : Int	Type : Int	Num : Double
1	Alice	500	4	800.2
2	Bob	1000	5	300.1
3	Chris	300	1	400.4
...	...	...	...	...
306	Quinne	800	2	40.6
307	Ray	300	3	200.5
308	Steve	200	3	600.6

Where are the blocks - Columnar (4KB blocks)

ID : Int	Name : Str(8)	Checking : Int	Type : Int	Num : Double
1	Alice	500	4	800.2
2	Bob	1000	5	300.1
3	Chris	300	1	400.4
...	...	...	...	...
4000	Walter	300	2	400.6
4001	Xavier	400	3	600.2
4002	Ygritte	200	3	800.9

Query for single record

Row Store:

- Find Block (Probably index)
- Pull block with record

Column Store:

- Find Block (Probably index)
- Pull Block for each Column

```
SELECT * FROM t WHERE id=700;
```

Aggregate Query

Row Store:

- Pull every block

Column Store:

- Pull blocks only for num

```
SELECT SUM(num) FROM t;
```

Single insert into table

Row Store:

- Insert tuple and shift rows

Column Store:

- Insert value and shift values for each column

Example Columnar Optimizations

- Compression
- Late Materialization

Compression

Data within a column is of the same type, which can improve data compression.

Compression can be further improved by sorting rows.

(Compression performs best on low-entropy, high-locality data.)

Compression: Run Length Encoding

WWWWWWWWWWWWB -> 12W1B

Compression: Run Length Encoding

- Sort first by columns with low cardinality

Name	Account Type	Bank Branch
Alice	Checking	Newton
Bob	Savings	Needham
Charlie	Savings	Wellesley
Dave	Checking	Wellesley
Evelyn	Checking	Newton

Compression: Run Length Encoding

- Sort by Account Type (2 options)

Name	Account Type	Bank Branch
Alice	Checking	Newton
Dave	Checking	Wellesley
Evelyn	Checking	Newton
Bob	Savings	Needham
Charlie	Savings	Wellesley

Compression: Run Length Encoding

- Sort by Bank Branch (3 options)

Name	Account Type	Bank Branch
Alice	Checking	Newton
Evelyn	Checking	Newton
Dave	Checking	Wellesley
Bob	Savings	Needham
Charlie	Savings	Wellesley

Compression: Run Length Encoding

- Compress!

WWWWWWWWWWWWB -> 12W1B

Name	Account Type	Bank Branch
1Alice1Evelyn1Dave1Bob1 Charlie	3Checking2Savings	2Newton1Wellesley1Needham1Wellesley

Late Materialization

- Often, queries want entities.
- You have columns.
- Try to keep columns separate as late as possible in query pipeline.

Advantages:

- Can avoid materializing some tuples
- Vectorized operations are fast on columns
- Keep other tools (compression)

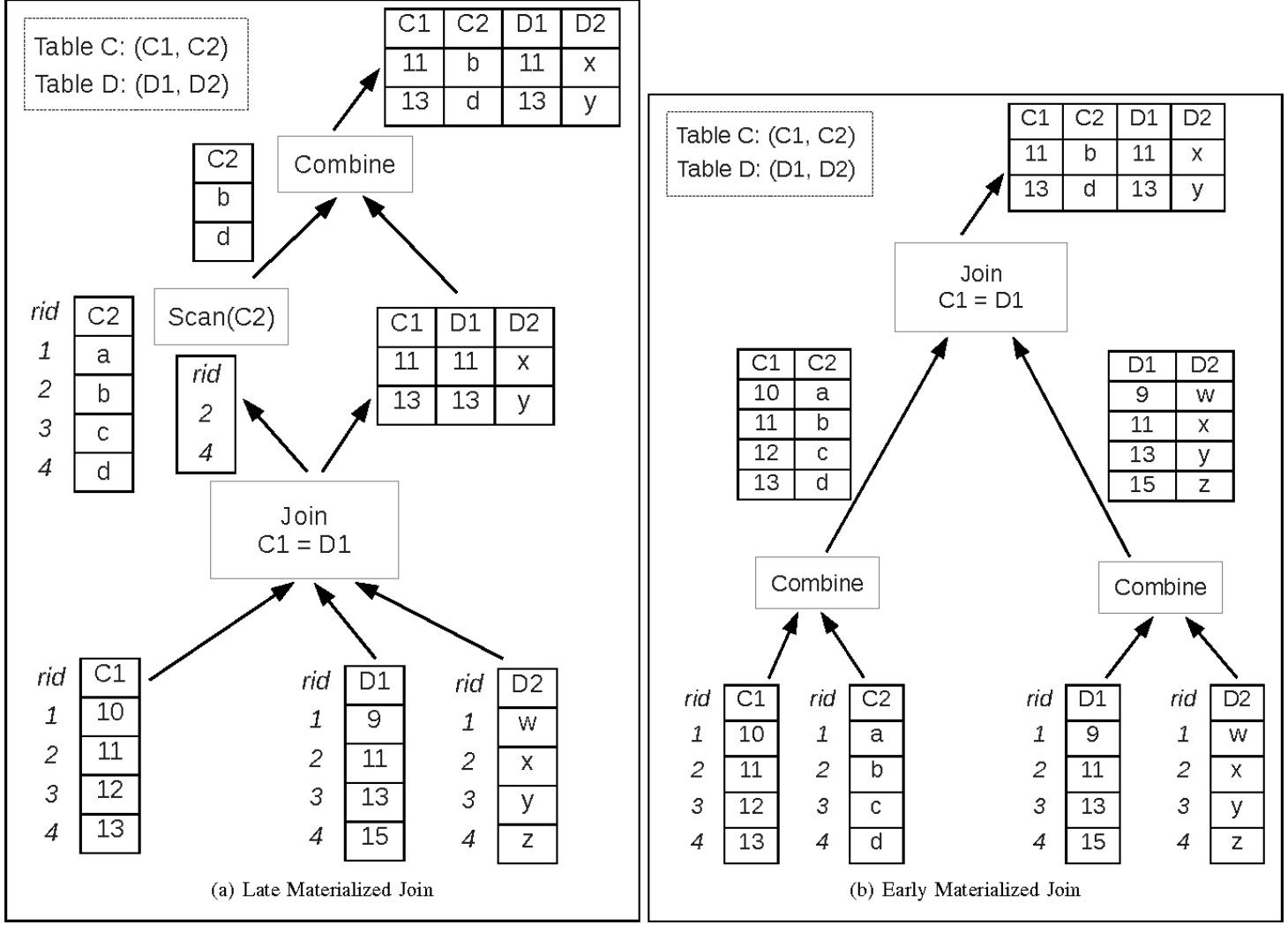


Fig. 2. Late and Early Materialized Joins in Vertica for the query "`SELECT C1,C2,D1,D2 FROM C,D WHERE C1=D1`"

DEMO?

~~DEMO?~~

NOPE.

Our (Forthcoming) Demo Implementation

Design:

- Based on HW4 Model
- File per column in relation
- Files can have different block sizes
  (Reflecting different types)
- Expose same CRU(D) operations

Quirks:

- Single Primary Key
- No sort keys
- No deletion

What's Great:

Our Implementation

Close to nothing.

What's frustrating:

Our Implementation

Almost everything

All the fiddlyness of HW4, again and again.

Lessons Learned:

Our Implementation

- Need to design for specific purpose to realize most gains.
- DBMS’s are messy, buy premade, don’t roll your own.
- Hopefully, some statistics about block pulls.

Common Use Cases (sweeping generalizations)

Row Stores:

- Often default
- Transactional situations (OLTP)
  - Small random writes
  - Often querying whole rows
  - Time efficient

Column Stores:

- Conscious choice
- Analytic Situations (OLAP)
  - Bulk writes
  - Often aggregate queries
  - Space efficient

Examples in the wild?

- **Amazon Redshift**
- **Google BigQuery**
- **Apache HBase**

“Big Data”

“Data Warehouses”

THANKS

Any Questions?