COLUMN-STORES VS. ROW-STORES: HOW DIFFERENT ARE THEY REALLY?

DANIEL J. ABADI (YALE)
SAMUEL R. MADDEN (MIT)
NABIL HACHEM (AVANTGARDE)

PRESENTATION BY PRANAV GOEL

Introduction

- On analytical workloads, Column store are found to perform order of magnitude better than traditional row-oriented database systems – “row stores”
- Elevator pitch: “column-stores are more I/O efficient for read-only queries since they only have to read from disk (or from memory) those attributes accessed by a query”
- Factor of 2 on price/performance
- Factor of 5 on performance

Data Warehouse and DBMS Software

- $5 Billion Data warehouse industry out of 20 Billion DBMS software industry of recurring revenues in 2010
- Growing at 8.5 % annually
- Total industry is expected to be at 105 Billion by 2020

Source Gartner

Motivation

Why not traditional DBMS system move to C-Store?

C-Store:

SAP HANA
Sybase IQ
Vertica

BIG QUESTIONS
Bringing life to debate

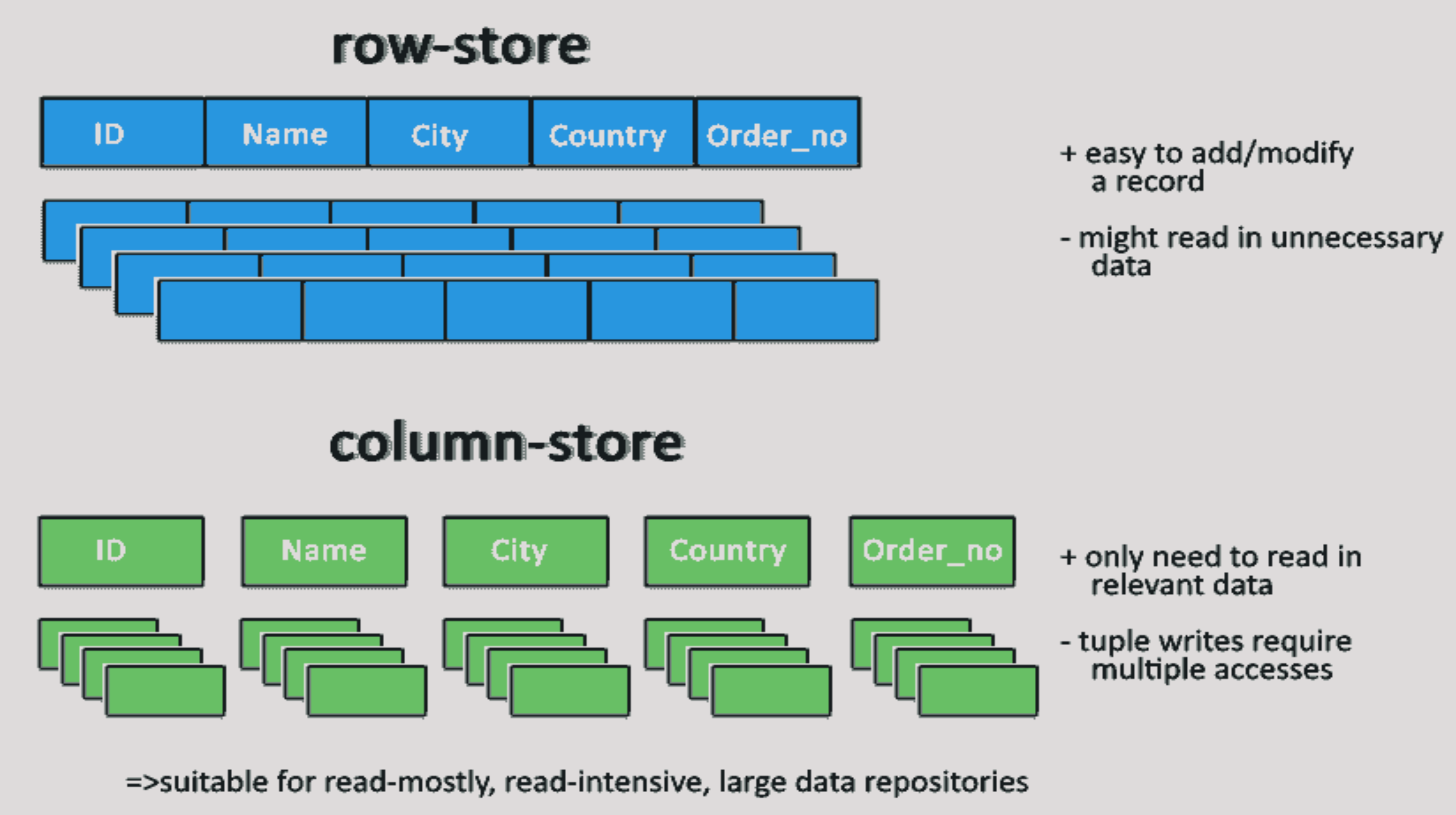
Key Questions:

- How much of the buzz around column- stores just marketing hype?
- Do you really need to buy products like SAP HANA or Vertica?
- How far will your current row-store take you?

Is this assumption valid?

- Can we adapt our row-store to get column-store performance?
- Can you simulate a column-store in a row-store?
- If not, what makes column-store so special?

Background



Source: Google Images

Background: Continued

Most of the queries do not process all the attributes of a particular relation.

- For example the query

```
Select c.name and c.address
From CUSTOMES as c Where c.region = 'Vancouver';
```

- Only processes three attributes of the relation CUSTOMER. But the customer relation can have more than three attributes.
- Column-stores are faster for OLAP operations and slower for OLTP operations with many row inserts

Paper Methodology

- Comparing row-store vs. column-store is dangerous/borderline meaningless
- Instead, compare row-store vs. row-store and column-store vs. column-store

    ✓ Simulate a column-store inside of a row-store

    ✓ Remove column-oriented features from column-store until it behaves like a row-store

Row Oriented Execution

- Common assumption would be to based on the storage layout, is to modify the row store physical structure in such a way that it behaves more like column store :

- The three ways to do these are:
  - ✓Vertical Partitioning
  - ✓Using index-only plans
  - ✓Using materialized views

First: Vertical Partitioning

- Idea: Vertical partition every relation

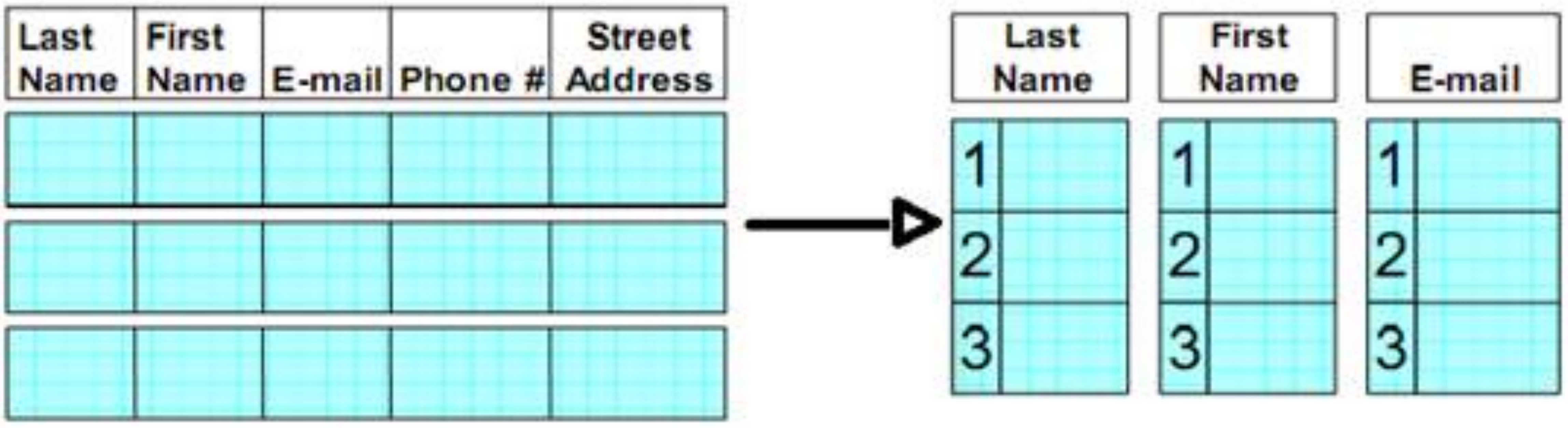
  - Mechanism to connect files from same row together :
    - This is done by addition integer ‘position’ column to every table
    - Primary keys are bad idea as they are could be long or composite
    - Thus each column have one Physical table.

  - Join are done based on position attributes when fetching multiple columns

Problems:

– “Position” - Space and disk bandwidth
– Header for every tuple – further space wastage

Vertical Partitioning : Example



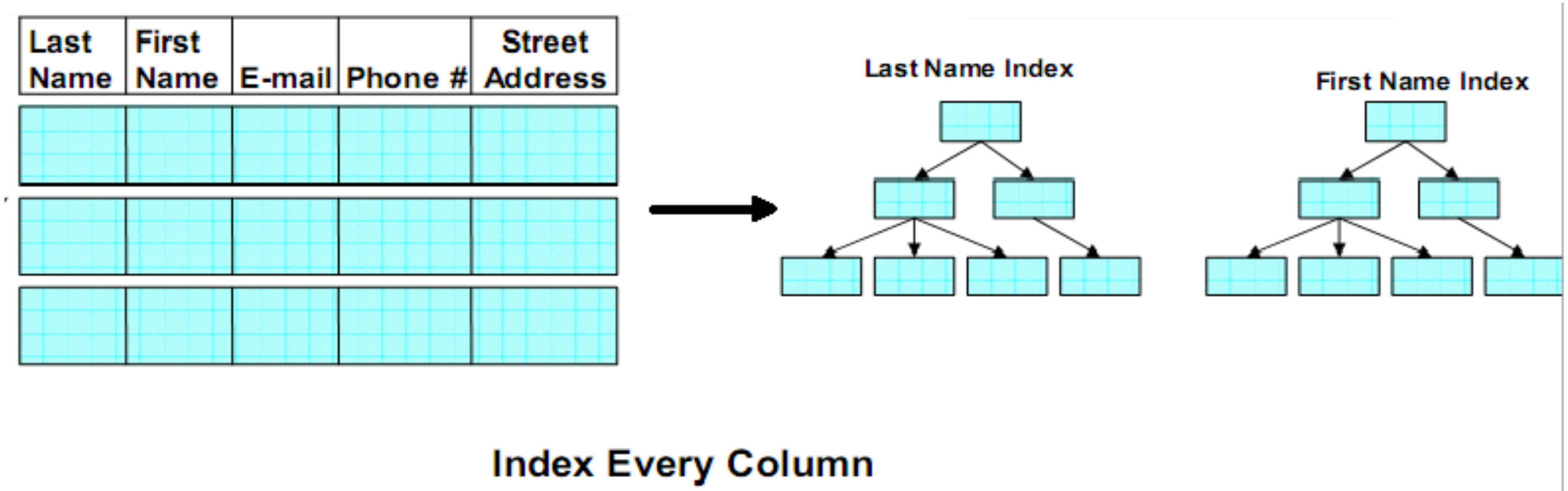
Vertical Partitioning

Google Images

Second: Index-Only Plans

- Approach
  - Create B+ tree like index for every column
  - Plans never access the actual tuples on disk
  - Tuple headers are not stored thus less overhead

Example:



Google Images

Index-Only Plans

Problems

- Separate indices may require full index scan, which is slower

Example:

SELECT AVG(salary) FROM emp WHERE age > 40

As there are separate indices on age and salary, an index only plan have to find ids corresponding to satisfying age and then merge with (id,salary) extracted from salary index

- Slow tuple construction

Third: Materialized Views(MV)

Methodology:

- Create an optimal set of materialized views for every query workload.

Idea behind Strategy:

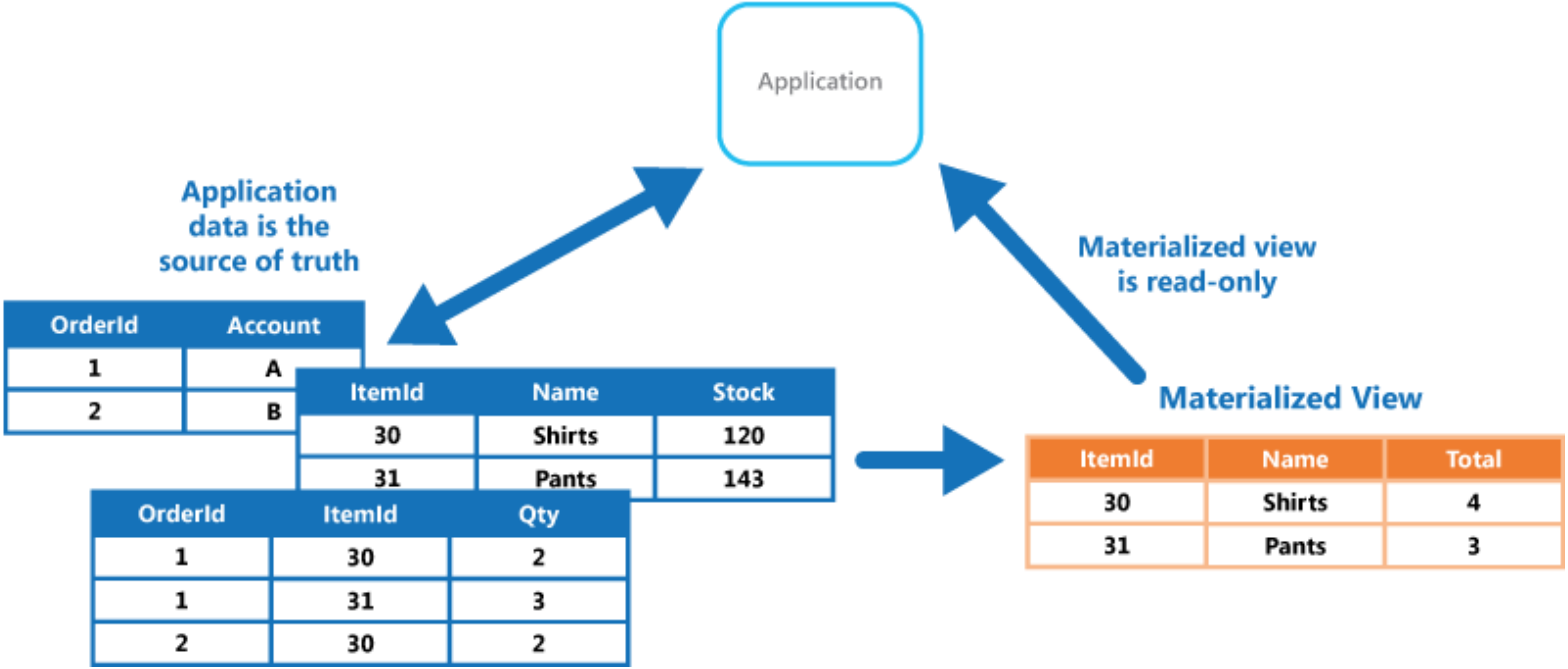
Only have columns needed for answer

Avoid overhead – perform better

Problems:

Advance knowledge of query workload thus practical in limited situations

Materialized Views: Example



OrderId	Account
1	A
2	B

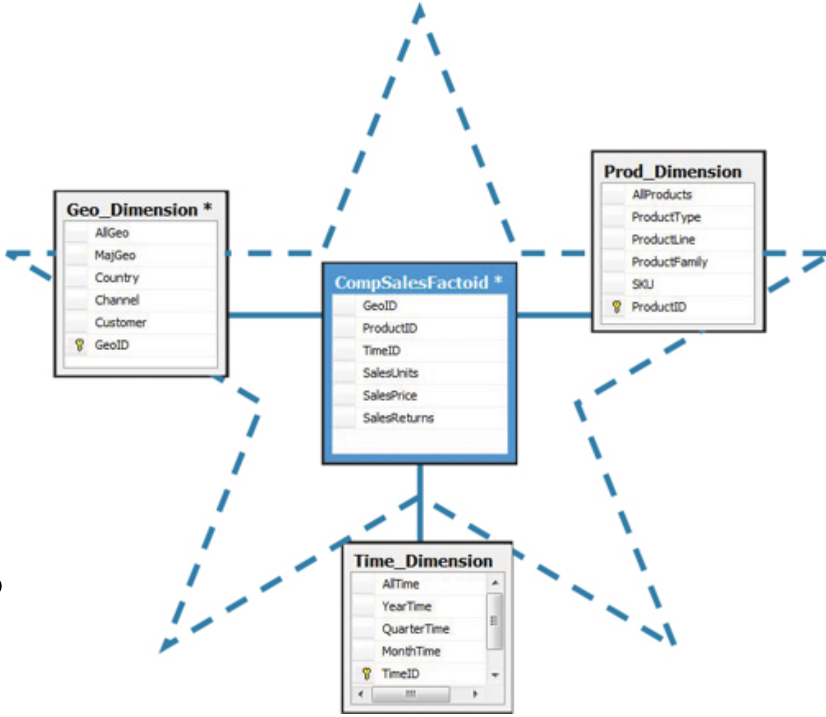
ItemId	Name	Stock
30	Shirts	120
31	Pants	143

OrderId	ItemId	Qty
1	30	2
1	31	3
2	30	2

ItemId	Name	Total
30	Shirts	4
31	Pants	3

Google Images

Benchmarking Methodology



Star Schema Benchmark (SSBM)

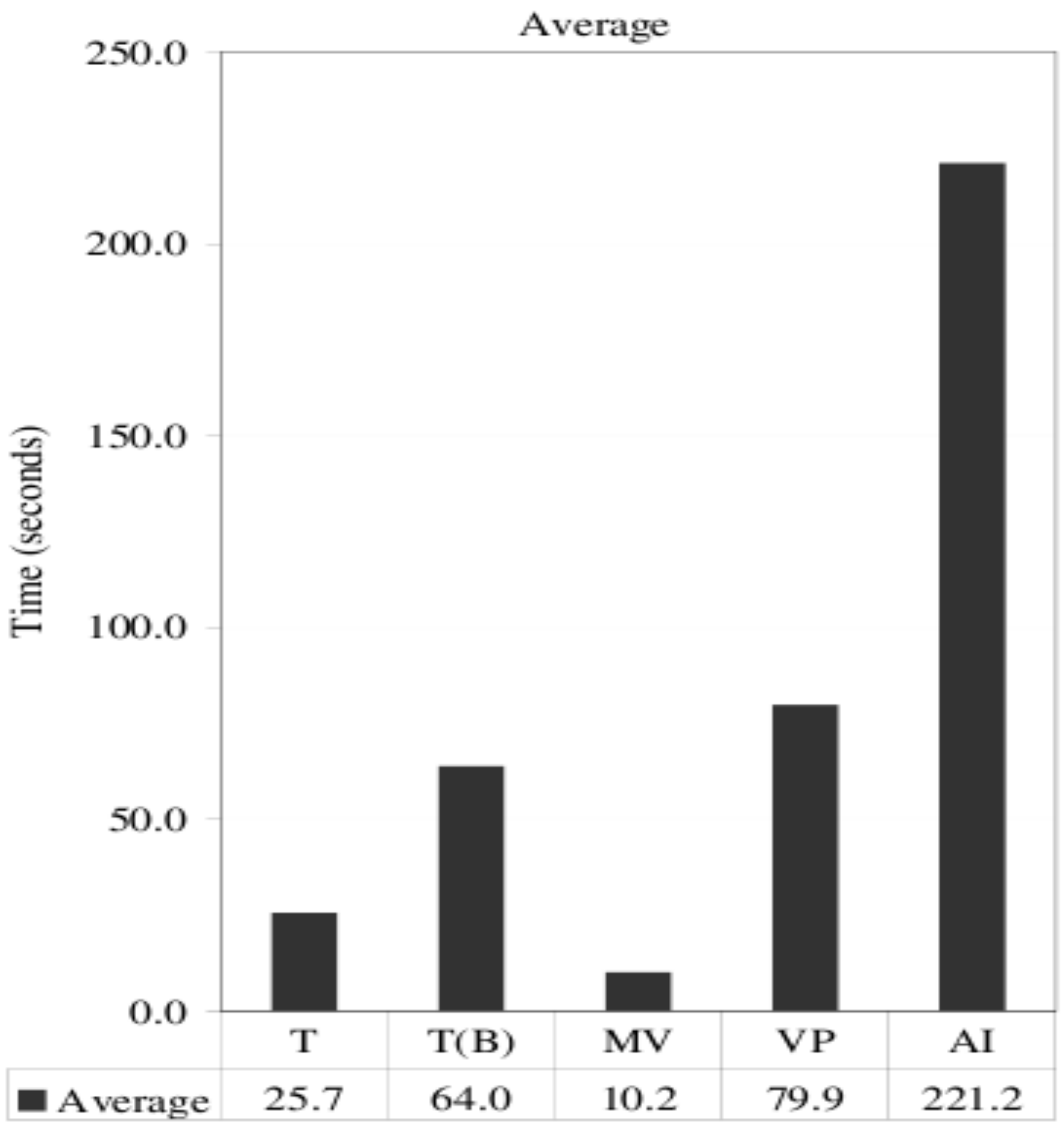
- Fact table contains 17 columns and 60,000,000 rows
- 4 dimension tables, biggest one has 80,000 rows
- Queries perform 2-4 joins between fact table and dimension tables, aggregate 1-2 columns from fact table
- All benchmarks are done on using similar hardware configuration named SYSTEM X for this paper

Performance Comparison Row Store

Average case performance across all workloads of the SSBM

T – Traditional
T(B) - Traditional Bitmap
MV – Materialized Views
VP – Vertical portioning
AI – All indexes

- MV performs best
- Index only are the worst



Column Store simulation in Row Store: Analysis

Vertical Partitioning:

- **Tuple overheads:**
  - LineOrder Table

    60 million tuples, 17 columns
  - 8 bytes of over head per row
  - 4 bytes of record-id

Tuple Header	TID	Column Data
xxxx	1	yyy
xxxx	2	yyy
xxx	3	yyy

	1 column (Compressed)	Whole table (Compressed)
Row Store	0.7 -1.1 GB	4 GB
Column Store	240 MB	2.3 GB

Column Store simulation in Row Store: Analysis

- All indexes approach is a poor way to simulate a column-store
- Problems with partitioning are fundamental – Store tuple header in a separate partition
    - Allow virtual TIDs
    - Combine clustered indexes, vertical partitioning

- What can possible be done to simulate column store in row store?
    - Need better support for partitioning at the storage layer
    - Need support for column-specific optimization at the executer level
    - Full integration: buffer pool, transaction manager

Column-Oriented Execution

- Different optimization for column oriented database –

  ✓ Compression

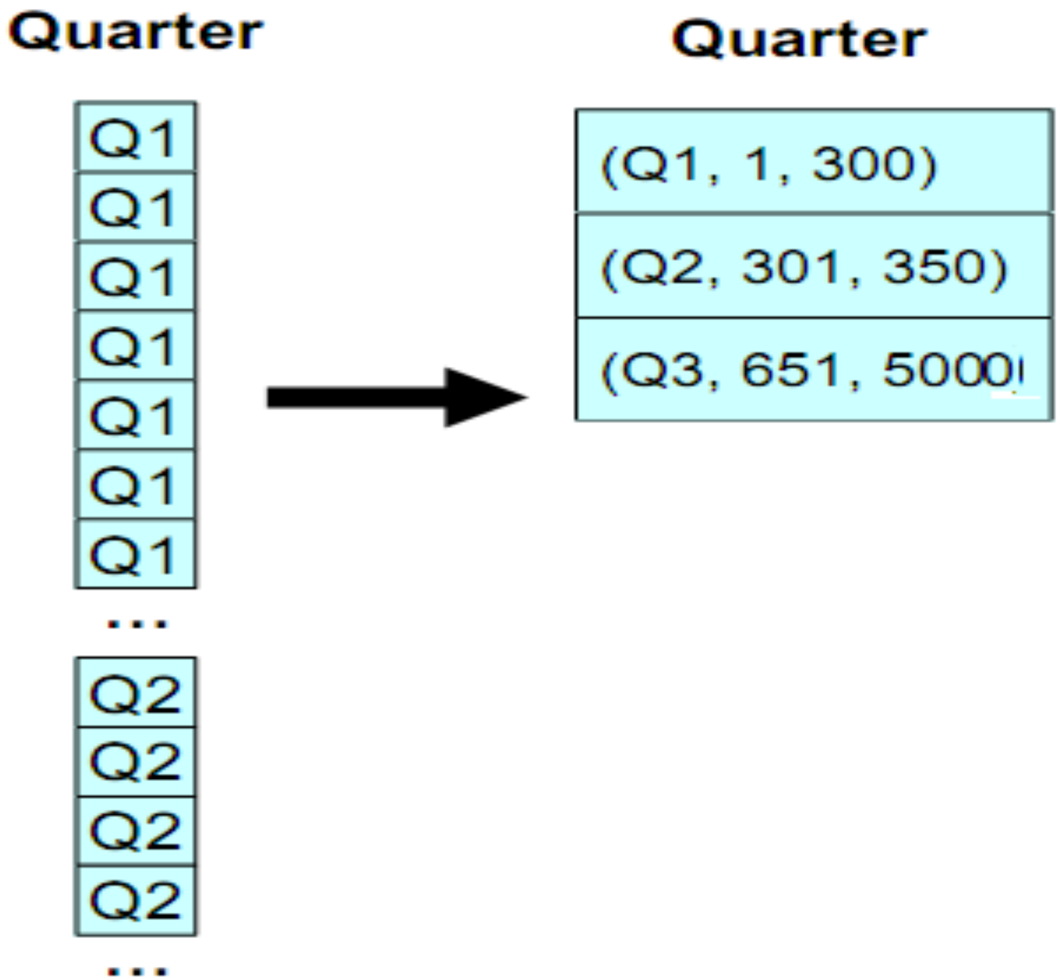
  ✓Late Materialization

  ✓ Block Iteration

  The idea is to minimize these advantages of column store to simulate row store in column store

Compression:

- Low information entropy (high data value locality) leads to high compression ratio
- Advantages
  - Disk Space is saved
  - Less I/O
  - CPU cost decrease if we can perform operation without decompressing



Idea: Use unsorted data

Late Materialization:

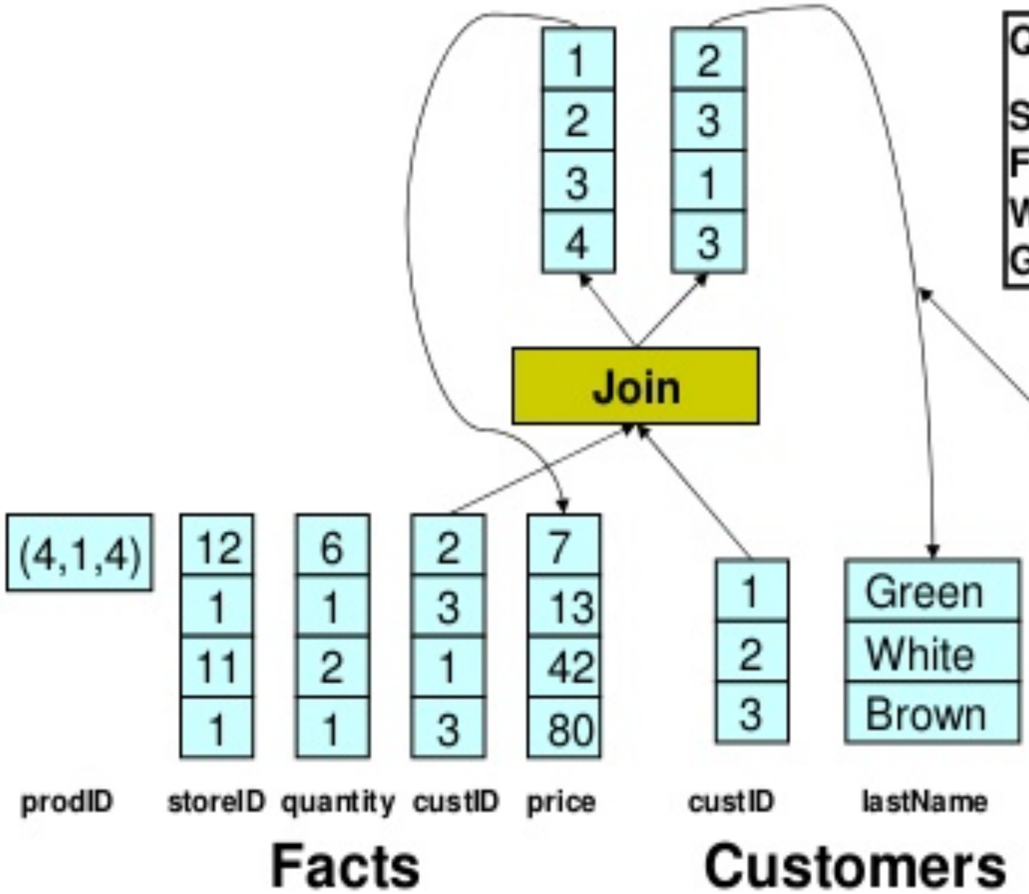
- Most queries require data from multiple columns that be combined together into rows to form information about an entity.
- Tuple Construction: Join like materialization of tuples
- Delay tuple construction

Advantages

- Unnecessary Construction of tuple is avoided
- Direct operations on compressed data
- Cache performance is improved

Idea: Use Early Materialization in the query plan

Late Materialization Example



QUERY:

SELECT C.lastName,SUM(F.price)
FROM facts AS F, customers AS C
WHERE F.custID = C.custID
GROUP BY C.lastName

Late materialized join causes out of order probing of projected columns from the inner relation

Block Iteration

Row Store first iterate through each tuple to extract needed attributes thus leading to tuple at a time processing (MySQL).

C-Store blocks of values from same column are sent to operator in single function call

- Iterate over blocks rather than tuples, like Batch Processing

Advantages:

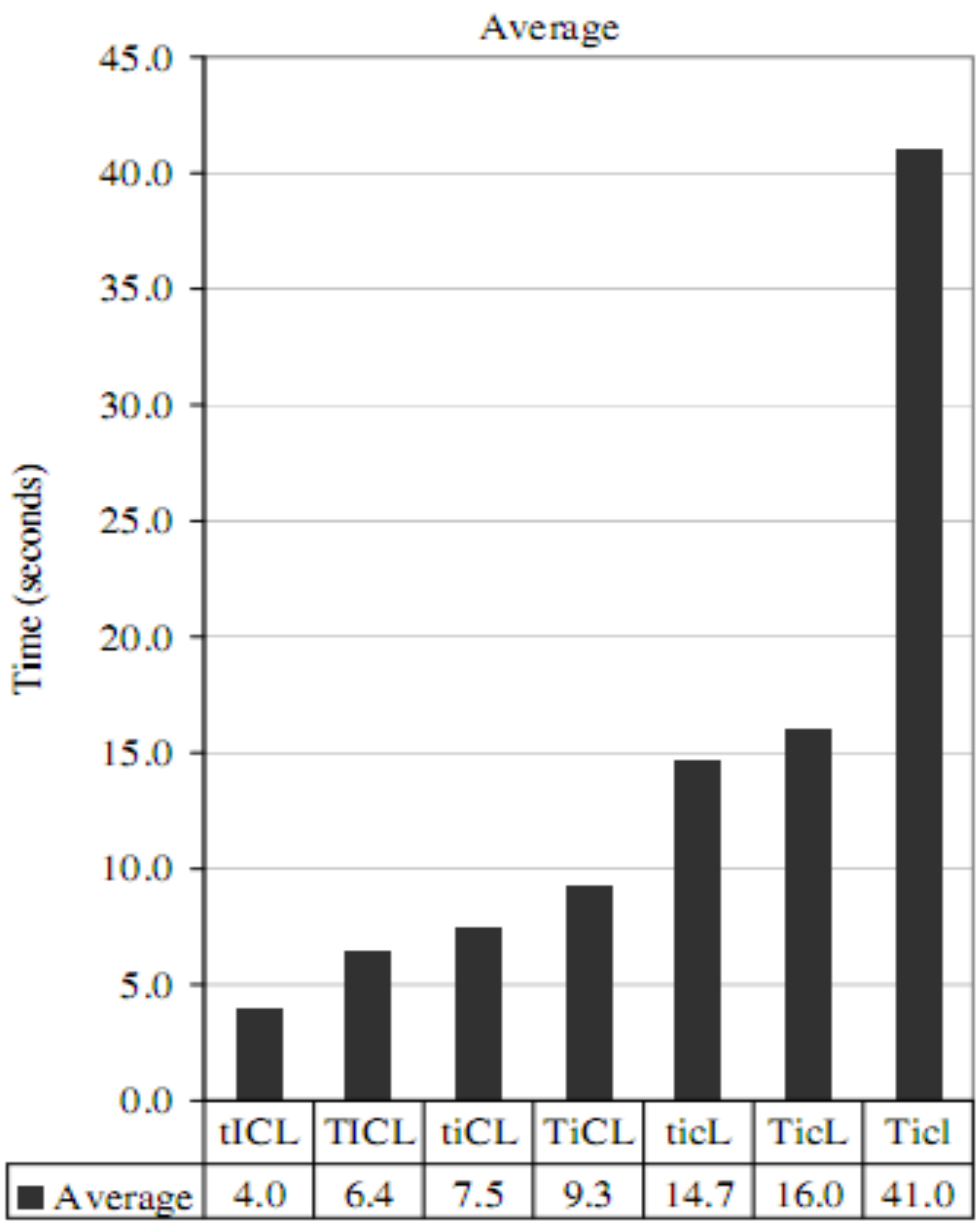
– Minimizes per-tuple overhead – Exploits potential for parallelism

Idea : Can be applied even in Row stores – IBM DB2 implements it

Experiment:

Performance of C store with various optimizations removed

T=tuple-at-a-time processing;
t=block processing;
I=invisible join enabled;
i=disabled;
C=compression enabled,
c=disabled;
L=late materialization enabled;
l=disabled;



CONCLUSION:

- TO SIMULATE COLUMN STORE IN ROW STORE, TECHNIQUES LIKE –

VERTICAL PARTITIONING , INDEX ONLY PLAN

DO NOT YIELD GOOD PERFORMANCE

HIGH PER-TUPLE OVERHEADS, HIGH TUPLE CONSTRUCTION COSTS ARE THE REASON

- WHERE AS IN COLUMN STORE –

LATE MATERIALIZING , COMPRESSION , BLOCK ITERATION

ARE THE REASONS FOR GOOD PERFORMANCE

MIGHT BE POSSIBLE TO SIMULATE A ROW-STORE IN A COLUMN-STORE,

NEED BETTER SUPPORT FOR VERTICAL PARTITIONING AT THE STORAGE LAYER

NEED SUPPORT FOR COLUMN-SPECIFIC OPTIMIZATIONS AT THE EXECUTER LEVEL

- DO WE SEE COLUMN STORE SIMULATION IN ROW STORE WORKING?

- WILL COLUMN STORE PROVIDE WRITE OPTIMIZATIONS COMPARABLE TO ROW STORES?

- USE ROW STORE AND COLUMN STORE INTERCHANGEABLY

  SAP HANA – USES CONCEPT OF DELTA MERGE

WHERE WRITES ARE DONE ON ROW STORE AND THEN MERGED WITH COLUMNS STORE

THANK YOU FOR
YOUR LİSTENİNG

DO YOU HAVE
ANY QUESTİONS?