

Advanced Topics in Distributed Systems

Philippe Cudré-Mauroux

September 2013

Erasmus Mundus Program, KTH--Sweden

Lecture 3 – Big Data Analytics

Why Data Warehouses?

- DBMSs designed to manage operational data
 - Goal: support every day activities
 - Online transaction processing (OLTP)
 - Ex: Tracking sales and inventory of each Wal-mart store
- Enterprises also need to analyze and explore their data
 - Goal: summarize and discover trends to support decision making
 - Online analytical processing (OLAP)
 - Ex: Analyzing sales of all Wal-mart stores by quarter and region
- To support OLAP consolidate all data into a **warehouse**
 - Lazy data replication

Data Warehouse Overview

- Consolidated data from many sources
 - Must create a single unified schema
 - The warehouse is like a materialized view
- Very large size: terabytes of data are common
- Complex read-only queries (no updates)
- Fast response time is important

Creating a Data Warehouse

- **Extract** data from distributed operational databases
- **Clean** to minimize errors and fill in missing information
- **Transform** to reconcile semantic mismatches
 - Performed by defining views over the data sources
- **Load** to materialize the above defined views
 - Build indexes and additional materialized views
- **Refresh** to propagate updates to warehouse periodically

Alternative: Distributed DBMS

- User submits a query at one site
- Query is defined over data located at different sites
 - Different physical locations
 - Different types of DBMSs
- Query optimizer finds the best distributed query plan
 - Query executes across all the locations
 - Results shipped to query site and returned to user

Distributed DBMS Limitations

- Top-down
 - Global, *a priori* data placement
 - Global query optimization
 - Distributed transactions, tight coupling
- Assumes full *cooperation* of all sites
- Assumes *uniform* sites
- Assumes *short-duration* operations
- *Limited scalability*

Back to Warehouses: Outline

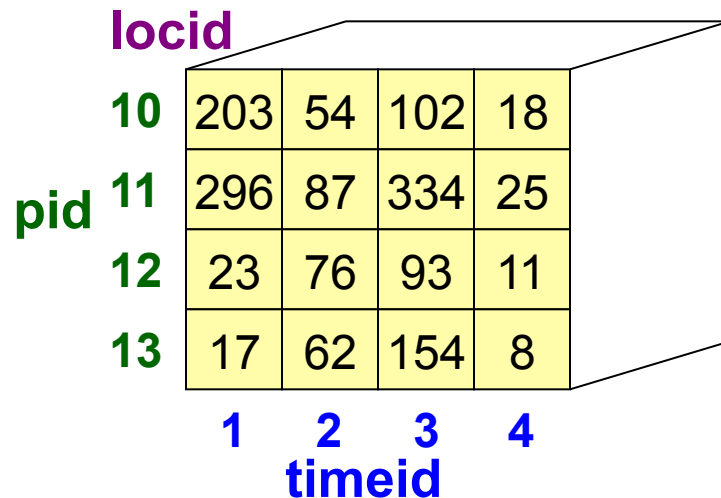
- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Data Analysis Cycle

- Formulate query that extracts data from the database
 - Typically ad-hoc complex query with group by and aggregate
- Visualize the data (e.g., spreadsheet)
 - Dataset is an N-dimensional space
- Analyze the data
 - Identify “interesting” subspace by aggregating other dimensions
 - Categorize the data and compare categories with each other
 - Roll-up and drill-down on the data

Multidimensional Data Model

- Focus of the analysis is a collection of **measures**
 - Example: Ikea sales
- Each measure depends on a set of **dimensions**
 - Example: **product** (pid), **location** (lid), and **time of the sale** (timeid)



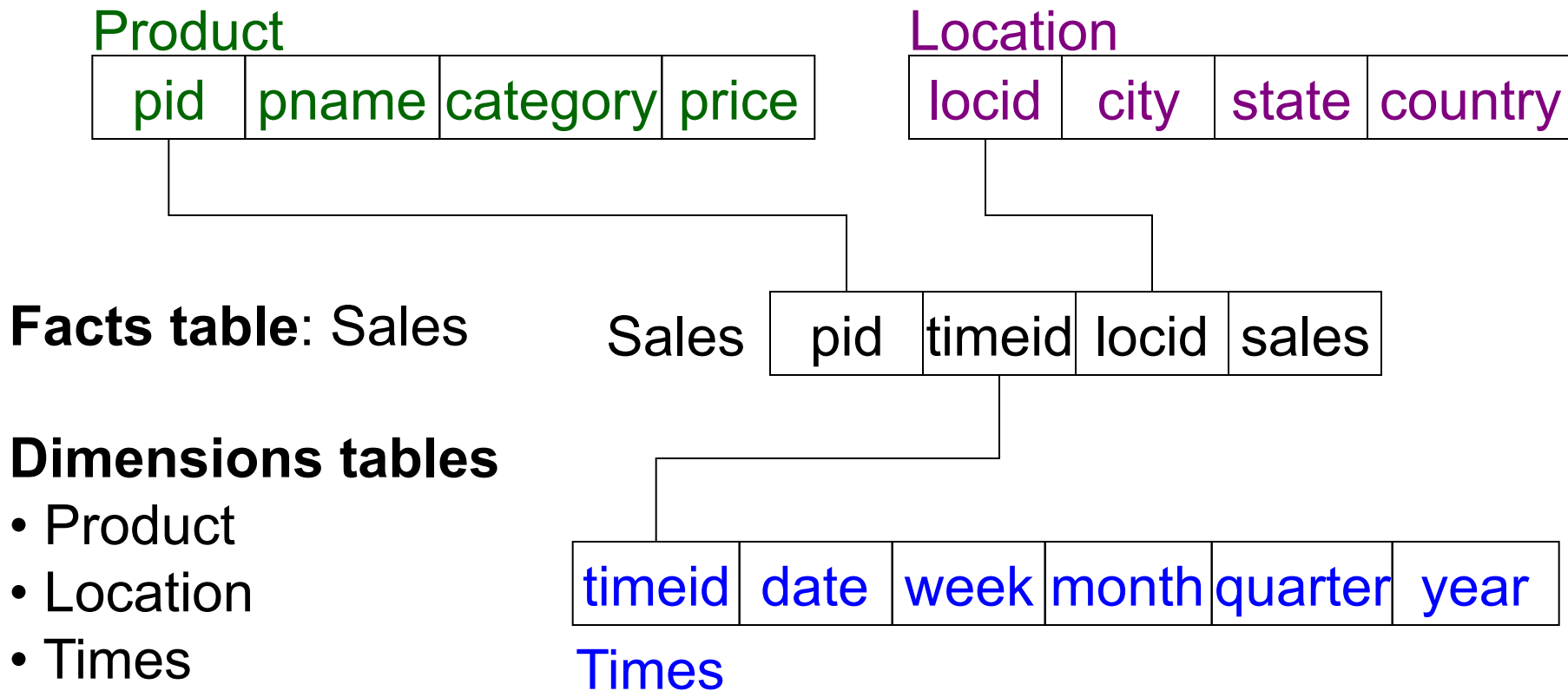
locid					
pid	10	203	54	102	18
	11	296	87	334	25
	12	23	76	93	11
	13	17	62	154	8
		1	2	3	4
		timeid			

Slicing: equality selection on one or more dimensions

Dicing: range selection

Star Schema

Representing multidimensional data as relations (ROLAP)



Dimension Hierarchies

Dimension values can form a hierarchy described by attributes

Product

category

pname

Time

year

quarter

week

month

date

Location

country

state

city

Desired Operations

- Histograms (agg. over computed categories)
 - Problem: awkward to express in SQL
- Summarize at different levels: **roll-up** and **drill-down**
 - Ex: total sales by day, week, quarter, and year

- **Pivoting**
 - **Creates table to quickly reorganize data**
 - Ex: pivot on location and time
 - Result of pivot is a **cross-tabulation**

	WI	CA	Total
2005	500	200	700
2006	150	850	1000
2007	250	400	650
Total	900	1450	2350

Challenge 1: Representation

- Problem: How to represent multi-level aggregation?
- Solution: special “all” value

T.year **L.state** **SUM(S.sales)**

2005	WI	500
2005	CA	200
2005	ALL	700
...
ALL	ALL	2350

Note: SQL-1999
standard uses NULL
values instead of ALL

Challenge 2: Computing Agg.

- Need 2^N different SQL queries to compute all aggregates (*powerset*)
 - Expressing roll-up and cross-tab queries is thus daunting
 - Cannot optimize all these independent queries
- Solution: CUBE and ROLLUP operators

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Data Cube

- CUBE is the N-dimensional generalization of aggregate

- Cube in SQL-1999

```
SELECT T.year, L.state, SUM(S.sales)
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid and S.locid=L.locid
GROUP BY CUBE (T.year,L.state)
```

- Creating a data cube requires generating the power set of the aggregation columns

Rollup

- Rollup produces a subset of a cube

- Rollup in SQL-1999

```
SELECT T.year, T.quarter, SUM(S.sales)
FROM Sales S, Times T
WHERE S.timeid=T.timeid
GROUP BY ROLLUP (T.year, T.quarter)
```

- Will aggregate over each pair of (year, quarter), each year, and total, but will **not** aggregate over each quarter

Computing Cubes and Rollups

- Naive algorithm
 - For each new tuple, update all matching cells
- More efficient algorithm
 - Use intermediate aggregates to compute others
 - Relatively easy for distributive and algebraic functions
- Updating a cube in response to updates is more challenging

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Indexes

- **Bitmap indexes:** good for sparse attributes (few values)

M	F
0	1
1	0
1	0

custid	name	gender	rating
10	Alice	F	3
11	Bob	M	4
12	Chuck	M	1

1	2	3	4
0	0	1	0
0	0	0	1
1	0	0	0

- **Join indexes:** to speed-up specific join queries
 - Example: Join fact table F with dimension tables D1 and D2
 - Index contain triples of rids $\langle r_1, r_2, r \rangle$ from D_1 , D_2 , and F that join

Materialized Views

- How to choose views to materialize?
 - Physical database tuning
- How to keep view up-to-date?
 - Could recompute entire view for each update: expensive
 - Better approach: incremental view maintenance
 - Example: recompute only affected partition
 - How often to synchronize? Periodic updates (at night) are typical

Outline

- Multidimensional data model and operations
- Data cube & rollup operators
- Data warehouse implementation issues
- Other extensions for data analysis

Additional Extensions for Decision Support

- Window queries

```
SELECT L.state, T.month, AVG(S.sales) over W AS movavg
FROM Sales S, Times T, Locations L
WHERE S.timeid = T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.State
              ORDER BY T.month
              RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
              AND INTERVAL '1' MONTH FOLLOWING)
```

- Top-k queries: optimize queries to return top k results
- Online aggregation: produce results incrementally

A Few Recent Systems

- Column Stores
 - Vertica / C-Store
- Main Memory Hybrid System
 - HyRise
- Array Analytics
 - SciDB

Why is evolving DBMSs so hard?

- Two fundamental problems to tackle
 - Obsolete **physical** model (impedance mismatch)
 - N-ary storage, relational tuples
 - B/R-trees
 - SPJ queries
 - ➡ Catastrophic performance in new application domains
 - Impractical **logical** guarantees
 - ACID properties
 - Brewer's conjecture (CAP theorem)
 - ➡ Inherent difficulty to scale-out

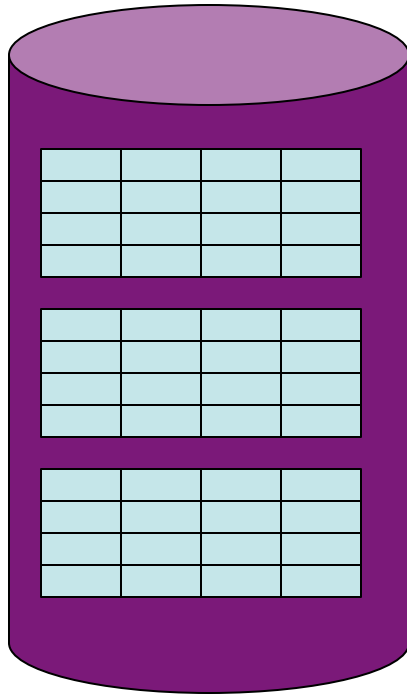
New Era of Data Management

- Users have gone away of DBMSs...
 - File-centric solutions
 - Procedural processing chains
- ... and have come back
 - Data abstractions (physical & logical)
 - Strong consistency guarantees

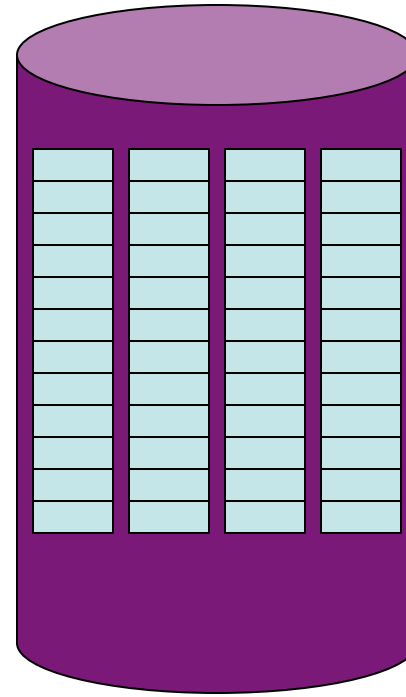
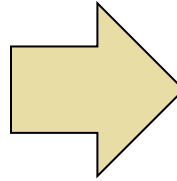
Column Store: Main Idea

- N-ary storage implies that all attributes are always read together
 - Cf. last lesson
- Data warehousing and business intelligence often require to analyze/aggregate attributes in isolation
 - Thus, would make sense to store each attribute separately!
- Simple idea but has profound consequences
 - Rewrite/rethink the whole database (incl. storage, indexes, optimizations, etc.)
 - Recent success stories (e.g., Vertica)

From Row-Store to Column-Store



Rows stored
contiguously on disk
(+ tuples headers)



Columns stored
contiguously on disk
(no headers needed)

Architecture

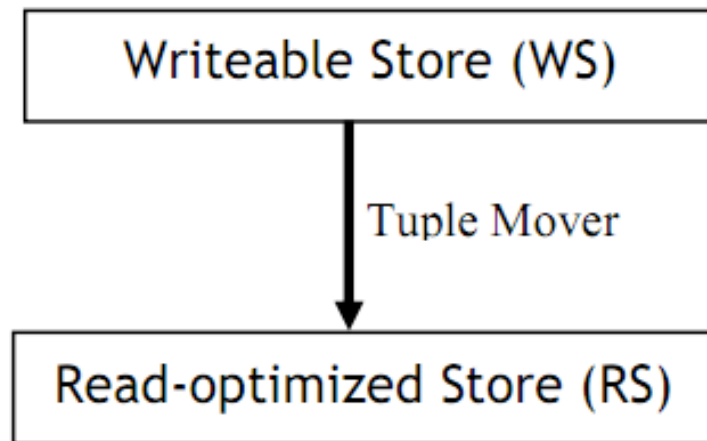


Figure 1. Architecture of C-Store

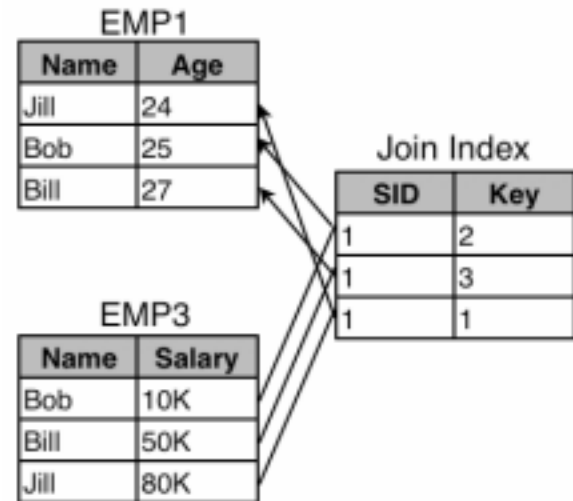


Figure 2: A join index from EMP3 to EMP1.

Vertical Storage

Row-based
(4 pages)

Page {

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

Column-based
(4 pages)

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

} Page

C-Store also
avoids large
tuple headers

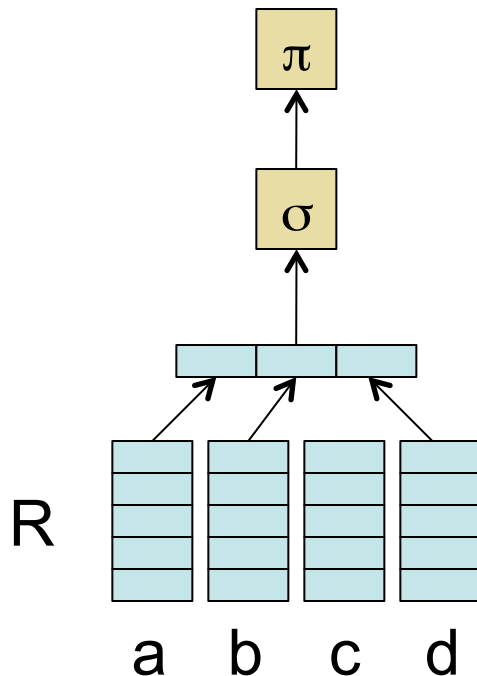
Column-Store Optimizations

- **Late tuple materialization** (3X improvement)
 - Process individual columns as long as possible
 - Merge columns into complete tuples as late as possible
- **Block iteration** (1.5X)
 - Pass blocks of values between ops instead of individual tuples
- **Compression**: e.g., run-length encoding of columns (10X)

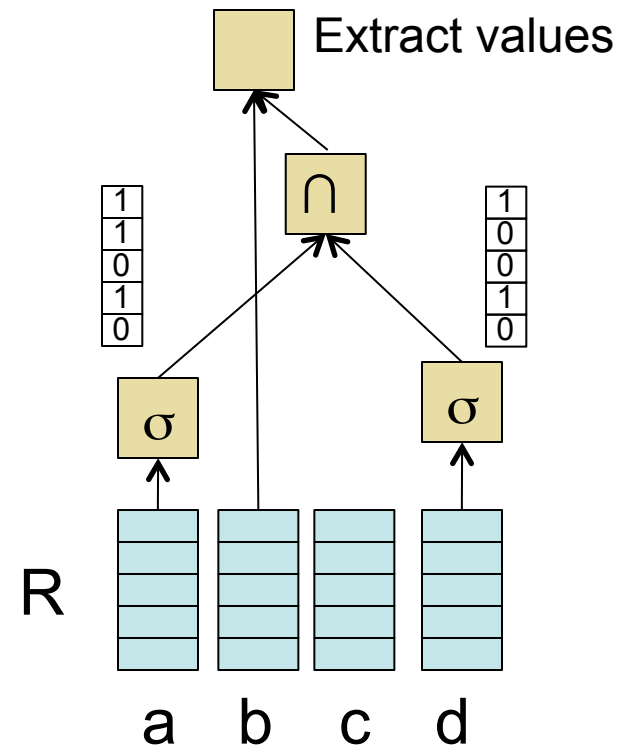
Late Tuple Materialization

Ex: SELECT R.b from R where R.a=X and R.d=Y

Early materialization



Late materialization



Compression Example

Row-based
(4 pages)

Column-based
(4 pages)

Compressed
(2 pages)

Page {

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

} Page

4XA	1X1
2XB	4X2
2XC	5X4

References

- Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross-Tab, and Sub-Totals. Jim Gray et. al. Data Mining and Knowledge Discovery 1, 29-53. 1997.
- C-Store: A Column Oriented DBMS. Mike Stonebreaker et al., VLDB 2005.