CompSci 516
Data Intensive Computing Systems

Lecture 22
Data Warehousing and Data Cube

Instructor: Sudeepa Roy

Announcements

- HW5 released (last one!)
  - Due on 04/20
  - No extension
- No class on Thursday 04/14
  - Make-up lecture on Monday 04/18, 4:30-5:45 pm, LSRC A247 (different room, same building, A-wing)
  - If you cannot make it – slides will be posted, or come to my office later
- We will do an optional review session before the final
  - will poll for topics to be reviewed on piazza and the date
  - material will be posted

Advanced Topics/Research Areas

- Lecture 21:
  - Datalog, NOSQL
- Lecture 22 (today 04/07):
  - Data Warehouse, OLAP, Data Cube
    - Pipehash algorithm at the end (from slide 54) is optional
- Lecture 23 (04/12, Tues):
  - Data Privacy
  - Guest lecture by Xi He
- Lecture 24 (04/18, Mon):
  - View selection
  - overview of Crowdsourcing in Databases, Data Integration, Data Cleaning, Incomplete Data and repairs, uncertain data, …
- Lecture 25 (04/19, Tues):
  - Data mining and association rule mining

Data Warehousing

Reading Material

- Optional:
  - (To be added)

Introduction

- Organizations analyze current and historical data
  - to identify useful patterns
  - to support business strategies
- Emphasis is on complex, interactive, exploratory analysis of very large datasets
- Created by integrating data from across all parts of an enterprise
- Data is fairly static
- Relevant once again for the recent "Big Data analysis"
  - to figure out what we can reuse, what we cannot

Three Complementary Trends

- Data Warehousing (DW):
  - Consolidate data from many sources in one large repository
  - Loading, periodic synchronization of replicas
  - Semantic integration
- OLAP:
  - Complex SQL queries and views.
  - Queries based on spreadsheet-style operations and “multidimensional” view of data.
  - Interactive and “online” queries.
- Data Mining:
  - Exploratory search for interesting trends and anomalies
  - Another lecture!

Data Warehousing

- A collection of decision support technologies
- To enable people in industry/organizations to make better decisions
  - Supports OLAP (On-Line Analytical Processing)
- Applications in
  - Manufacturing
  - Retail
  - Finance
  - Transportation
  - Healthcare
  - …
- Typically maintained separately from "Operational Databases"
  - Operational Databases support OLTP (On-Line Transaction Processing)

OLTP	Data Warehousing/OLAP
Applications: Order entry, sales update, banking transactions	Applications: Decision support in industry/organization
Detailed, up-to-date data	Summarized, historical data (from multiple operational db, grows over time)
Structured, repetitive, short tasks	Query intensive, ad hoc, complex queries
Each transaction reads/updates only a few tuples (tens of)	Each query can accesses many records, and perform many joins, scans, aggregates
Important: Consistency, recoverability, Maximizing transaction throughput	Important: Query throughput Response times

Data Marts

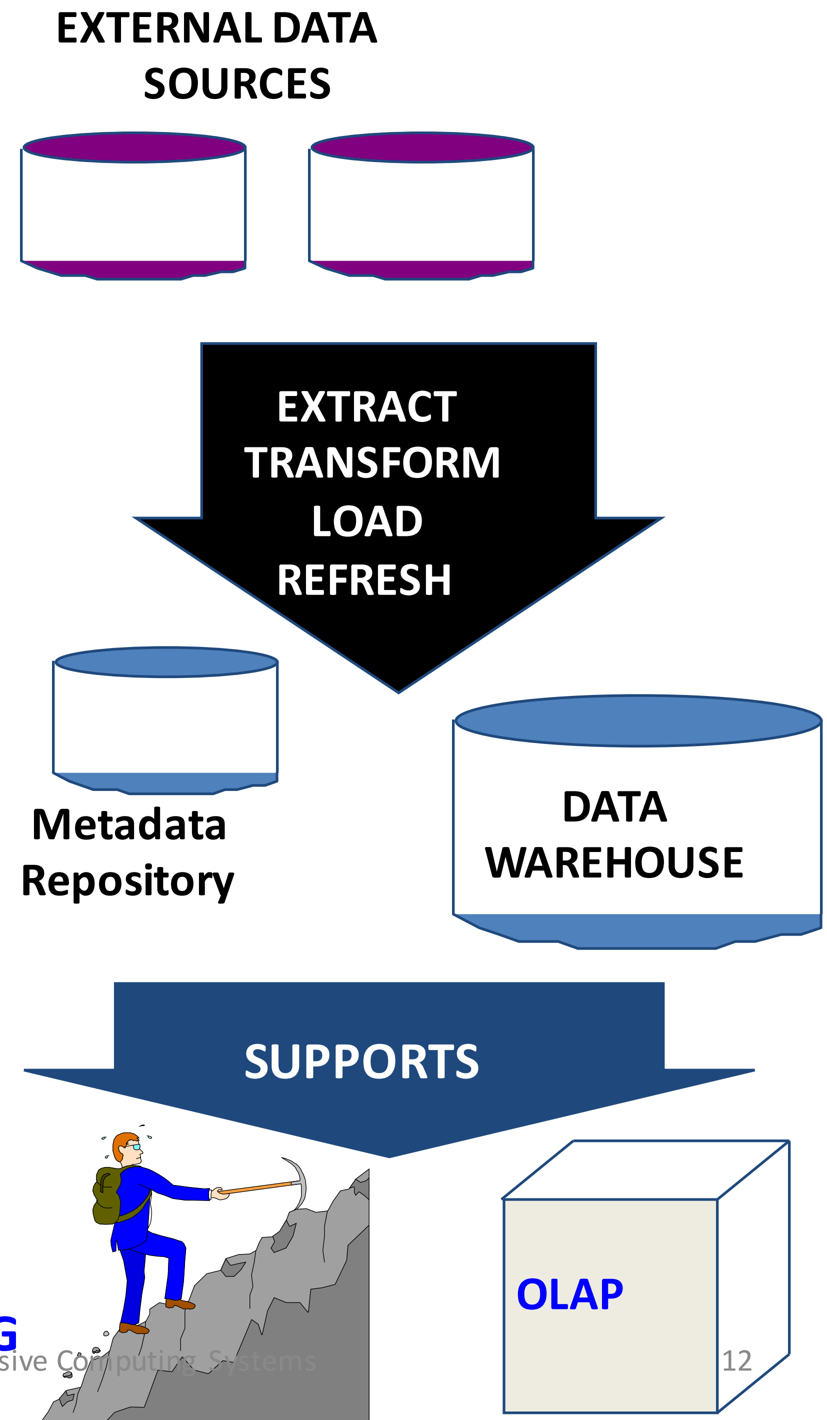
- Data marts
  - subsets of data on selected subjects
  - e.g. Marketing data mart can include customer, product, sales
  - Department-focused, no enterprise-wide consensus needed
  - But may lead to complex integration problems in the long run

ROLAP and MOLAP

- Relational OLAP (ROLAP)
  - On top of standard relational DBMS
  - Data is stored in relational DBMS
  - Supports extensions to SQL to access multi-dimensional data
- Multidimensional OLAP (MOLAP)
  - Directly stores multidimensional data in special data structures (e.g. arrays)

Data Warehousing to Mining

- Integrated data spanning long time periods, often augmented with summary information
- Several gigabytes to terabytes common
- Interactive response times expected for complex queries; ad-hoc updates uncommon



Warehousing Issues

- Semantic Integration: When getting data from multiple sources, must eliminate mismatches
  - e.g., different currencies, schemas
- Heterogeneous Sources: Must access data from a variety of source formats and repositories
  - Replication capabilities can be exploited here
- Load, Refresh, Purge: Must load data, periodically refresh it, and purge too-old data
- Metadata Management: Must keep track of source, loading time, and other information for all data in the warehouse

DW Architecture

- Extract data from multiple operational DB and external sources
- Clean/integrate/transform/store
- refresh periodically
  - update base and derived data
  - admin decides when and how
- Main DW and several data marts (possibly)
- Managed by one or more servers and front end tools
- Additional meta data and monitoring/admin tools

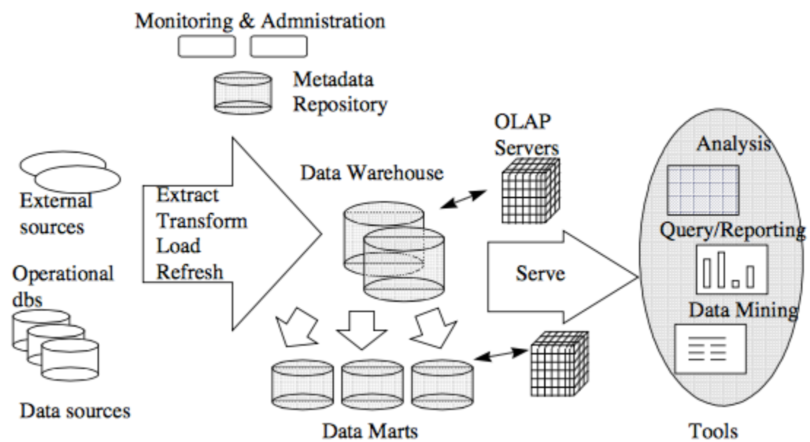


Figure 1. Data Warehousing Architecture

ROLAP: Star Schema

- To reflect multi-dimensional views of data
- Single fact table
- Single table for every dimension
- Each tuple in the fact table consists of
  - pointers (foreign key) to each of the dimensions (multi-dimensional coordinates)
  - numeric value for those coordinates
- Each dimension table contains attributes of that dimension

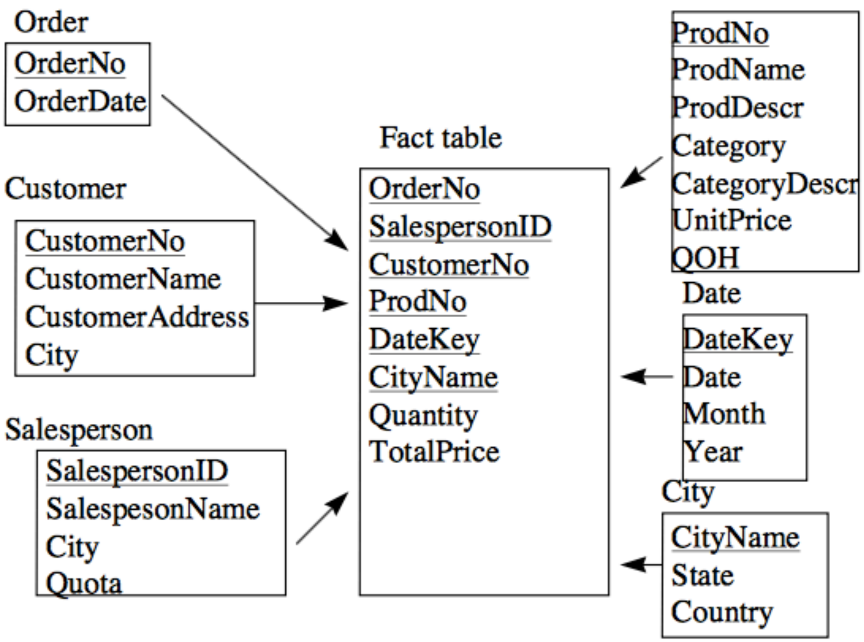


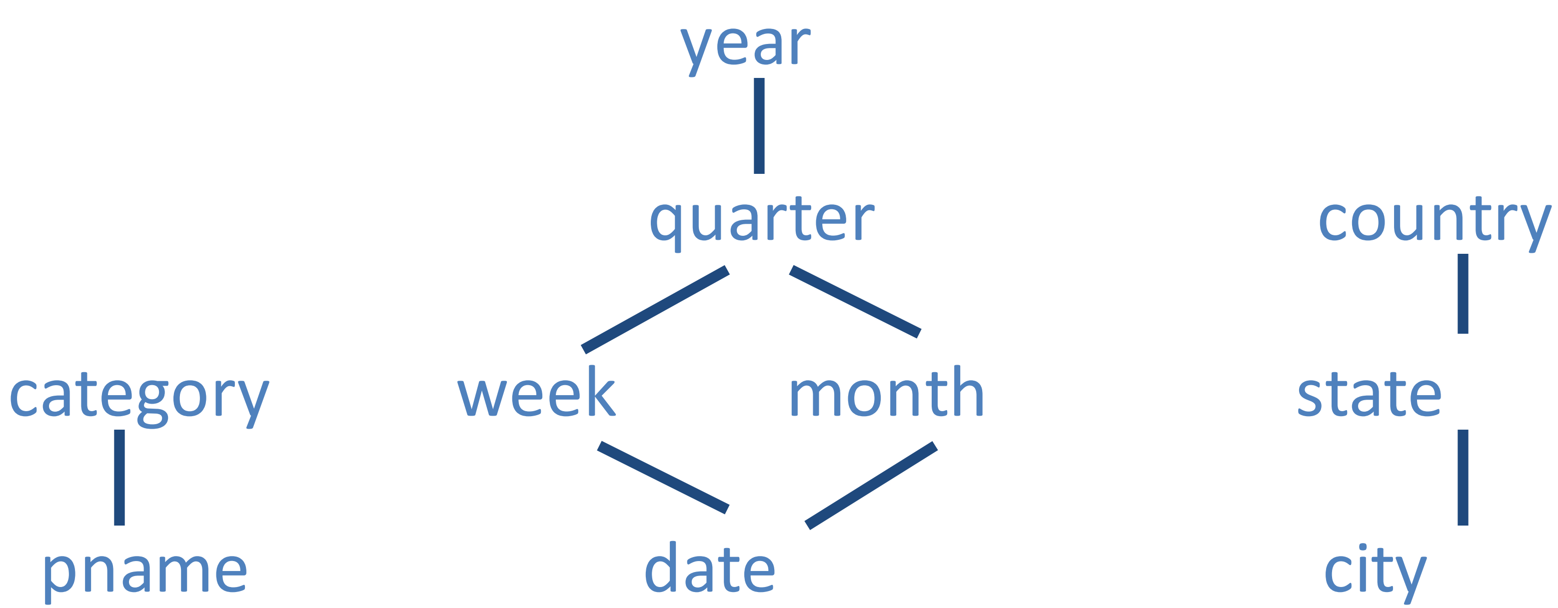
Figure 3. A Star Schema.

No support for attribute hierarchies

Dimension Hierarchies

- For each dimension, the set of values can be organized in a hierarchy:

PRODUCT **TIME** **LOCATION**



ROLAP: Snowflake Schema

- Refines star-schema
- Dimensional hierarchy is explicitly represented

- (+) Dimension tables easier to maintain
  - suppose the "category description is being changed
- (-) De-normalized structure
  - may be easier to browse
- Fact Constellations
  - Multiple fact tables share some dimensional tables
  - e.g. Projected and Actual Expenses may share many dimensions

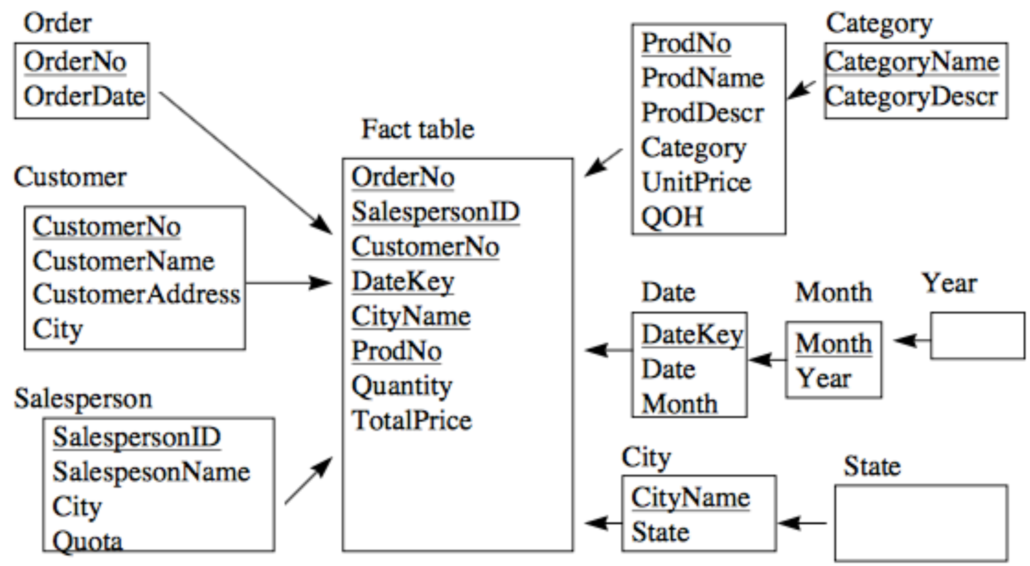


Figure 4. A Snowflake Schema.

OLAP Queries

- Influenced by SQL and by spreadsheets.
- A common operation is to aggregate a measure over one or more dimensions.
  - Find total sales.
  - Find total sales for each city, or for each state.
  - Find top five products ranked by total sales.
- Roll-up: Aggregating at different levels of a dimension hierarchy.
  - E.g., Given total sales by city, we can roll-up to get sales by state.

OLAP
and
Data Cube

Sales (Model, Year, Color, Units)

Motivation: OLAP Queries

- Data analysts are interested in exploring trends and anomalies
  - Possibly by visualization (Excel) - 2D or 3D plots
  - "Dimensionality Reduction" by summarizing data and computing aggregates
  - Influenced by SQL and by spreadsheets.
  - A common operation is to aggregate a measure over one or more dimensions.
- Find total unit sales for each
  1. Model
  2. Model, broken into years
  3. Year, broken into colors
  4. Year
  5. Model, broken into color, ….

Sales (Model, Year, Color, Units)

Naïve Approach

Run a number of queries

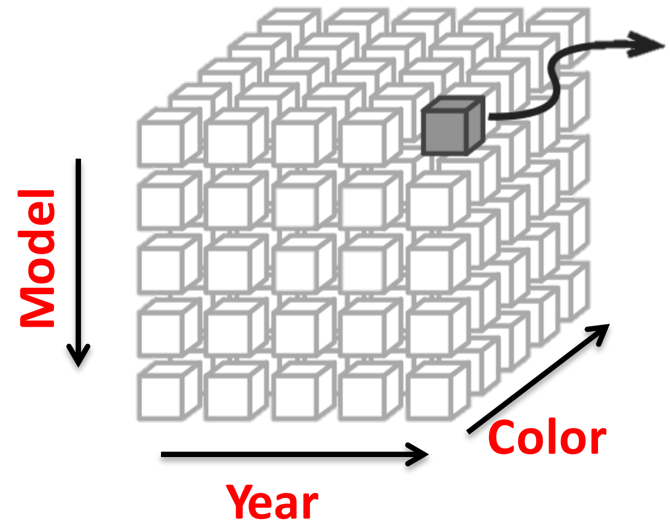
```
SELECT sum(units)
FROM Sales

SELECT Color, sum(units)
FROM Sales
GROUP BY Color

SELECT Year, sum(units)
FROM Sales
GROUP BY Year

SELECT Model, Year, sum(units)
FROM Sales
GROUP BY Model, Year
….
```

Total Unit sales



- Data cube generalizes Histogram, Roll-Ups, Cross-Tabs
- More complex to do these with GROUP-BY

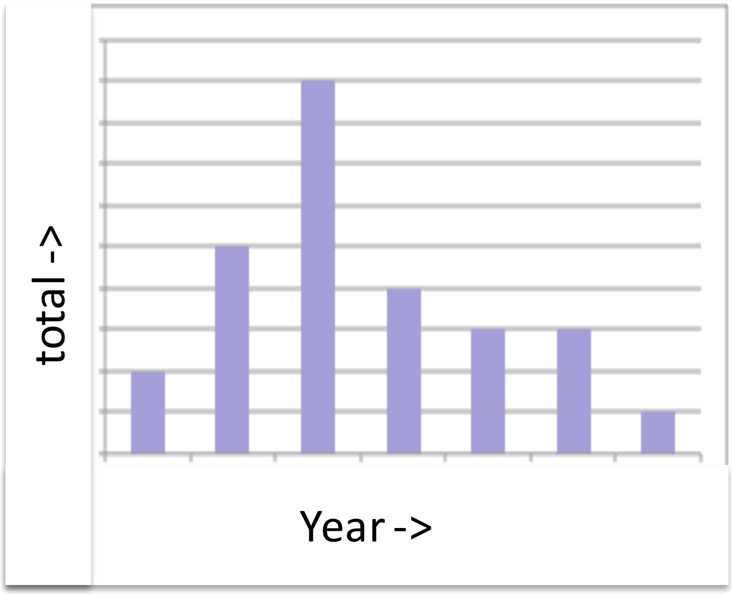
- How many sub-queries?
- How many sub-queries for 8 attributes?

Sales (Model, Year, Color, Units)

Histograms

A tabulated frequency of computed values

```
SELECT Year, COUNT(Units) as total
FROM Sales
GROUP BY Year
ORDER BY Year
```

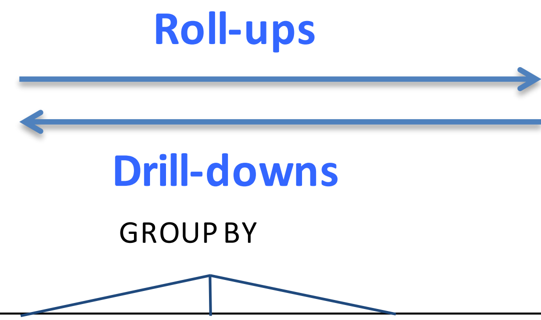


May require a nested SELECT to compute

Sales (Model, Year, Color, Units)

Roll-Ups

- Analysis reports start at a coarse level, go to finer levels
- Order of attribute matters
- Not relational data (empty cells no keys)



Model	**Year**	**Color**	**Model, Year, Color**	**Model, Year**	**Model**
Chevy	1994	Black	50		
Chevy	1994	White	40		
				90	
Chevy	1995	Black	115		
Chevy	1995	White	85		
				200	
					290

Sales (Model, Year, Color, Units)

Roll-Ups

- Another representation (Chris Date'96)
- Relational, but
  - long attribute names
  - hard to express in SQL and repetition

GROUP BY

Model	Year	Color	Model, Year, Color	Model, Year	Model
Chevy	1994	Black	50	90	290
Chevy	1994	White	40	90	290
Chevy	1995	Black	85	200	290
Chevy	1995	Black	115	200	290

Sales (Model, Year, Color, Units)

'ALL' Construct

Easier to visualize roll-up if allow ALL to fill in the super-aggregates

```
SELECT Model, Year, Color, SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
         GROUP BY Model, Year, Color
UNION
SELECT Model, Year, 'ALL', SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
     GROUP BY Model, Year
UNION
…
UNION
SELECT 'ALL', 'ALL', 'ALL', SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy';
```

Model	Year	Color	Units
Chevy	1994	Black	50
Chevy	1994	White	40
Chevy	1994	'ALL'	90
Chevy	1995	Black	85
Chevy	1995	White	115
Chevy	1995	'ALL'	200
Chevy	'ALL'	'ALL'	290

Sales (Model, Year, Color, Units)

Traditional Roll-Up

Model	**Year**	**Color**	**Model, Year, Color**	**Model, Year**	**Model**
Chevy	1994	Black	50		
Chevy	1994	White	40		
				90	
Chevy	1995	Black	115		
Chevy	1995	White	85		
				200	
					290

‘ALL’ Roll-Up

Model	**Year**	**Color**	**Units**
Chevy	1994	Black	50
Chevy	1994	White	40
Chevy	1994	‘ALL’	90
Chevy	1995	Black	85
Chevy	1995	White	115
Chevy	1995	‘ALL’	200
Chevy	‘ALL’	‘ALL’	290

- Roll-ups are asymmetric

Sales (Model, Year, Color, Units)

Cross Tabulation

If we made the roll-up symmetric, we would get a cross-tabulation

Generalizes to higher dimensions

```
SELECT Model, 'ALL', Color, SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
     GROUP BY Model, Color
```

Chevy	1994	1995	Total (ALL)
Black	50	85	135
White	40	115	155
Total (ALL)	90	200	290

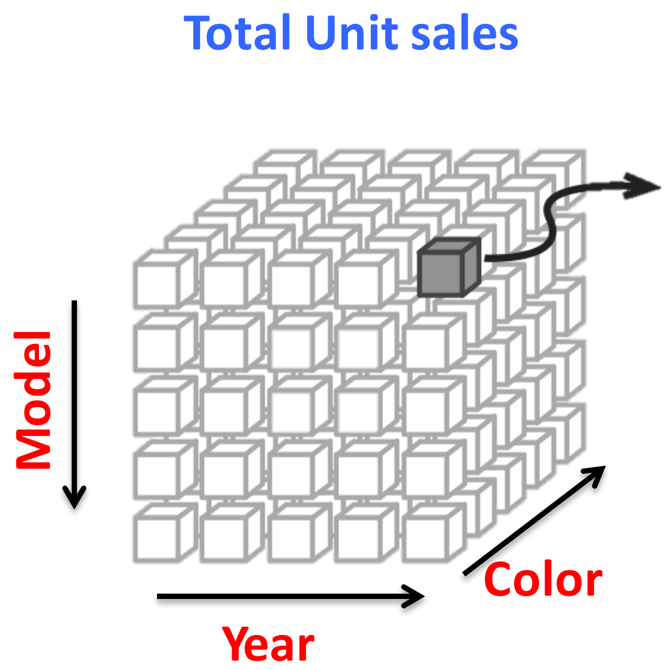
Is the problem solved with Cross-Tab and GROUP-BYs with 'ALL'?

- Requires a lot of GROUP BYs (64 for 6-dimension)
- Too complex to optimize (64 scans, 64 sort/hash, slow)

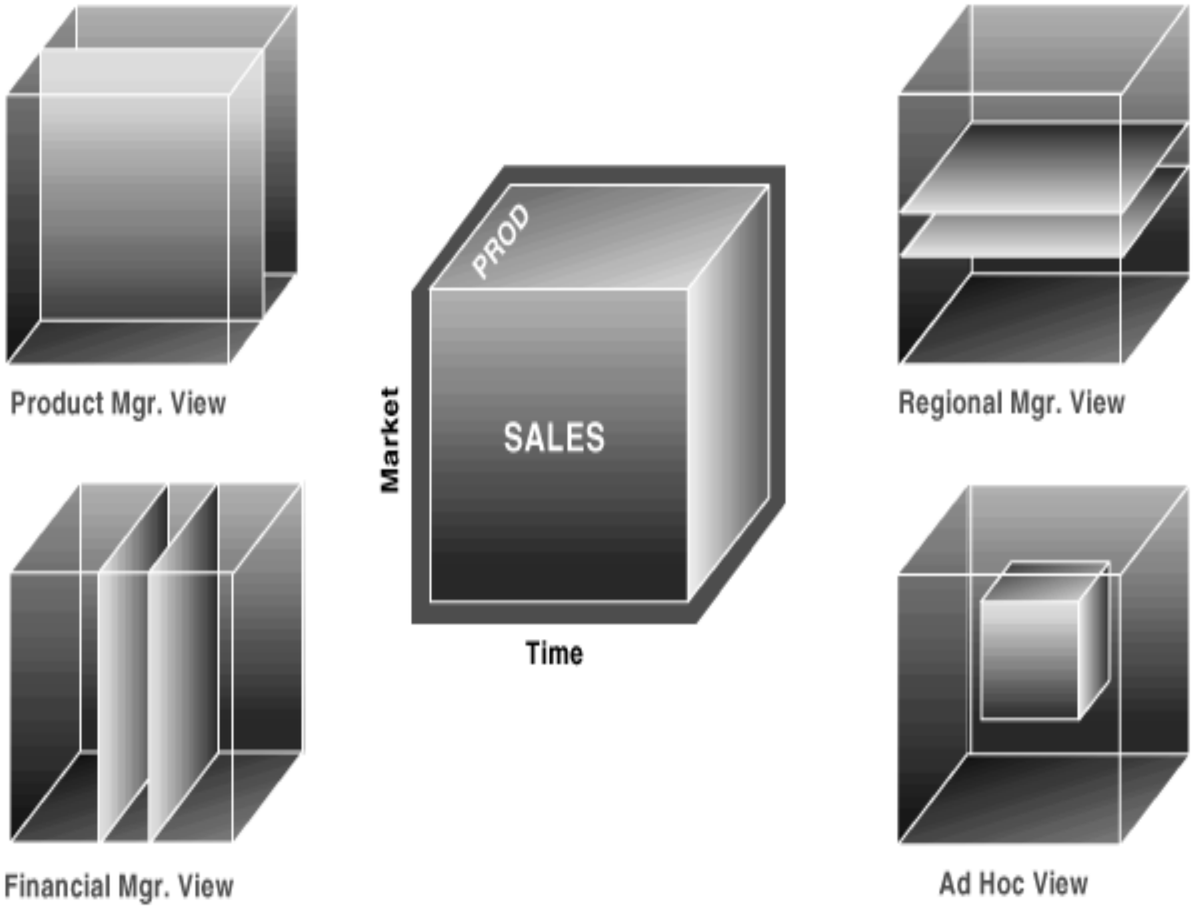
Sales (Model, Year, Color, Units)

Data Cube: Intuition

```
SELECT 'ALL', 'ALL', 'ALL', sum(units)
FROM Sales
UNION
SELECT 'ALL', 'ALL', Color, sum(units)
FROM Sales
GROUP BY Color
UNION
SELECT 'ALL', Year, 'ALL', sum(units)
FROM Sales
GROUP BY Year
UNION
SELECT Model, Year, 'ALL', sum(units)
FROM Sales
GROUP BY Model, Year
UNION
….
```



Data Cube



Ack: from slides by Laurel Orr and Jeremy Hyrkas, UW

Data Cube

- Computes the aggregate on all possible combinations of group by columns.
- If there are N attributes, there are 2^N-1 super-aggregates.
- If the cardinality of the N attributes are C_1,..., C_N, then there are a total of $(C_1+1)...(C_N+1)$ values in the cube.
- ROLL-UP is similar but just looks at N aggregates

Sales (Model, Year, Color, Units)

Data Cube Syntax

- SQL Server

```
SELECT Model, Year, Color, sum(units)
FROM Sales
GROUP BY Model, Year, Color
WITH CUBE
```

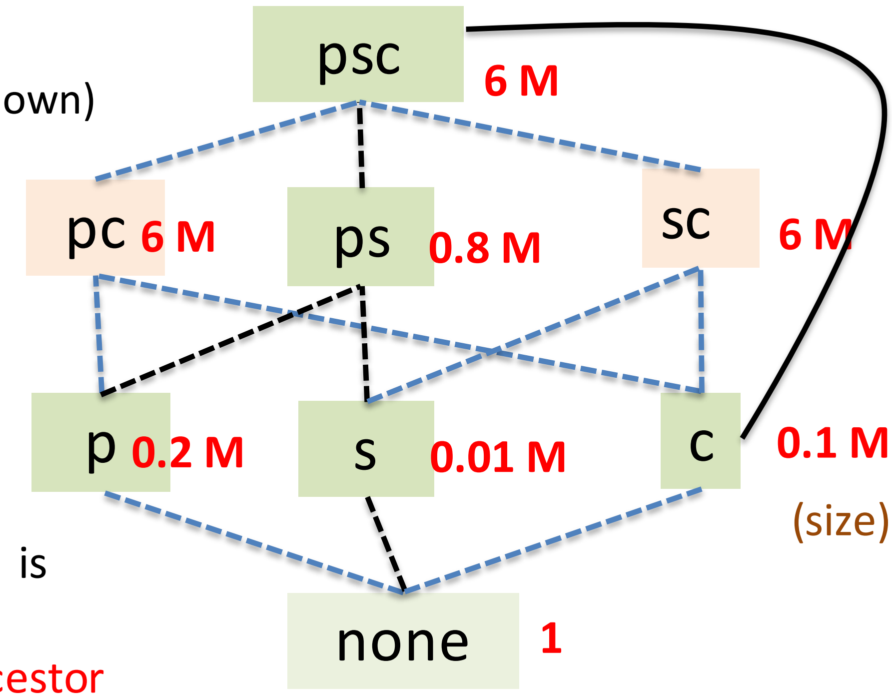
Types of Aggregates

- Distributive: input can be partitioned into disjoint sets and aggregated separately
  - COUNT, SUM, MIN
- Algebraic: can be composed of distributive aggregates
  - AVG
- Holistic: aggregate must be computed over the entire input set
  - MEDIAN

- Efficient computation of the CUBE operator depends on the type of aggregate
  - Distributive and Algebraic aggregates motivate optimizations

"Lattice" Framework for Data Cube

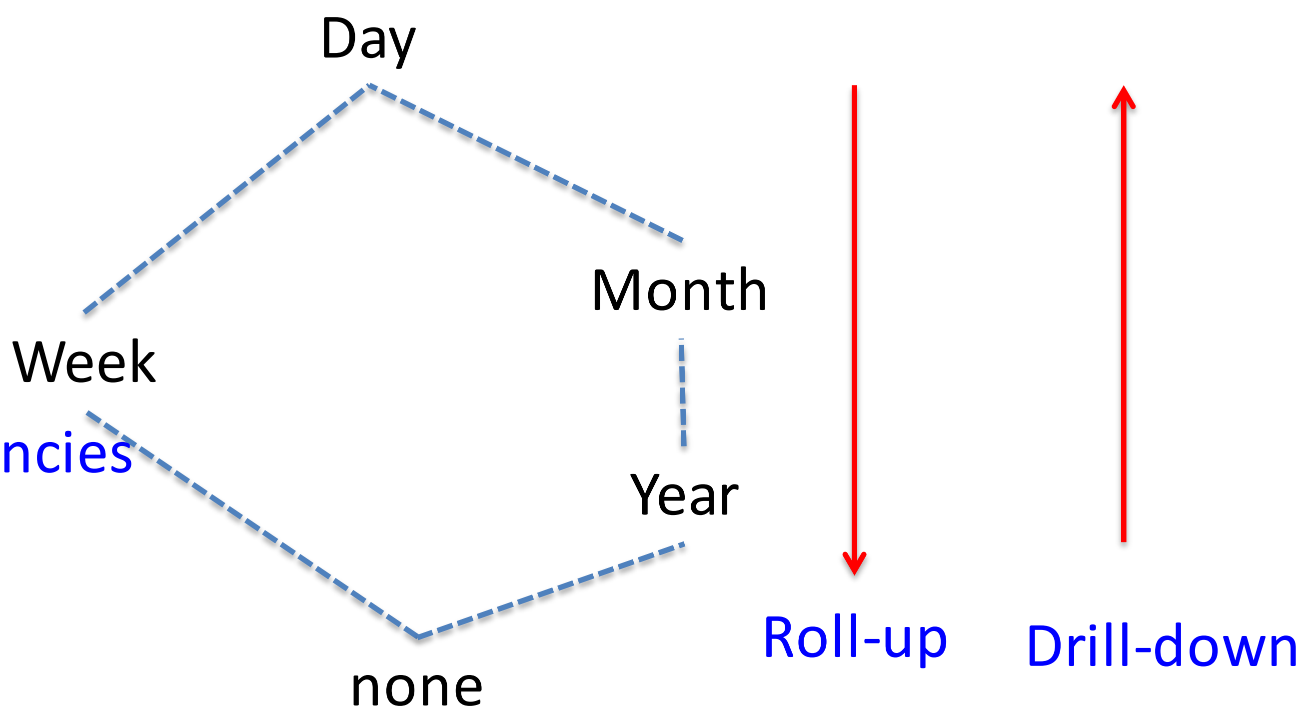
- Can model group-by queries well
  - Users typically go along the edges
  - Drill-down (going up) and Roll-up (going down) along a path
- The order of "materializing views":
  - Suppose a set S of views has to be materialized
  - We do not need to go to raw data to materialize every view
  - "Topological order" sort in S (first all ancestors are materialized, then a node is materialized)
  - Then, materialize from the smallest ancestor
  - e.g. materializing s from ps needs to read 0.8 M, but from sc needs to read 6M tuples
- However, further consideration:
  - sorted order of ancestors
  - pipeline or not – see the pipesort algo
- More on View selection in Lecture 24



Hierarchies

- Some dimensions (attributes) are organized in hierarchies
- Should be considered while deciding materialization of views

Hierarchy of time attributes



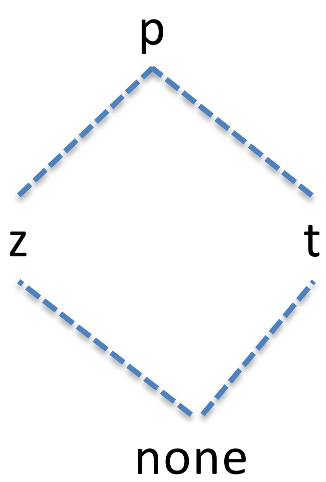
Functional dependencies

Month -> Year

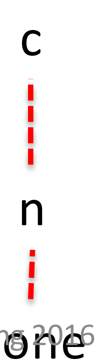
(Jan’15) -> 2015

Combining Two Hierarchical Dimensions

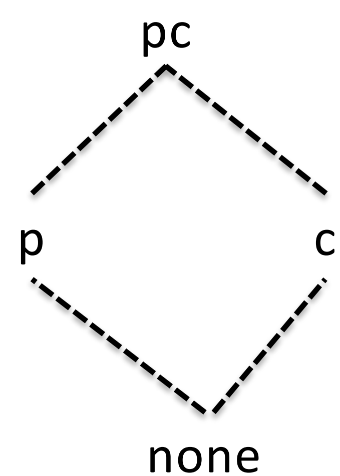
part (p):
size(z), type(t)



customer (c):
nation(n)

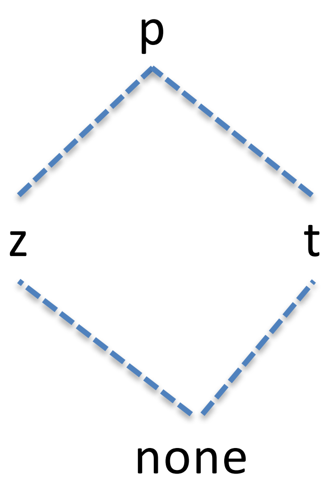


- lattice structure between part (p) and customer (c) without hierarchy
- How can we extend the lattice structure to include the hierarchy for parts + hierarchy for customers?
- Solution: use product lattice

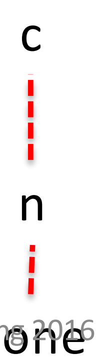


Combining Two Hierarchical Dimensions

part (p):
size(z), type(t)

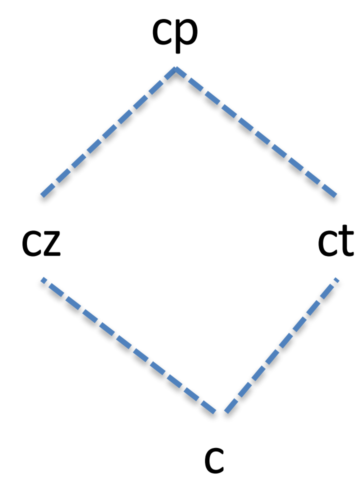


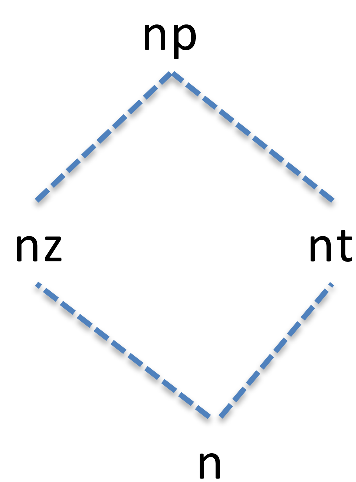
customer (c):
nation(n)

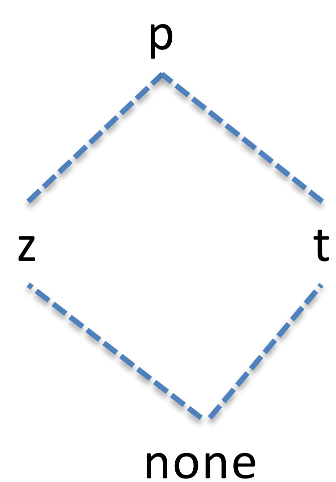


Direct Product Lattice

- Select one lattice, say for p
- Combine it with each value in the hierarchy of c

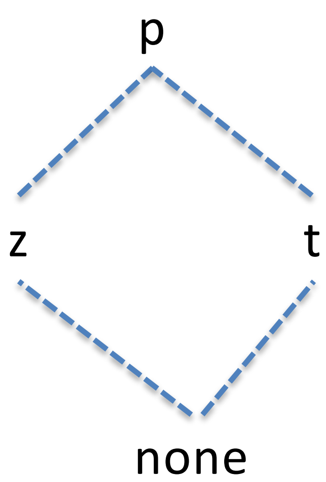




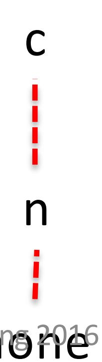


Combining Two Hierarchical Dimensions

part (p):
size(z), type(t)

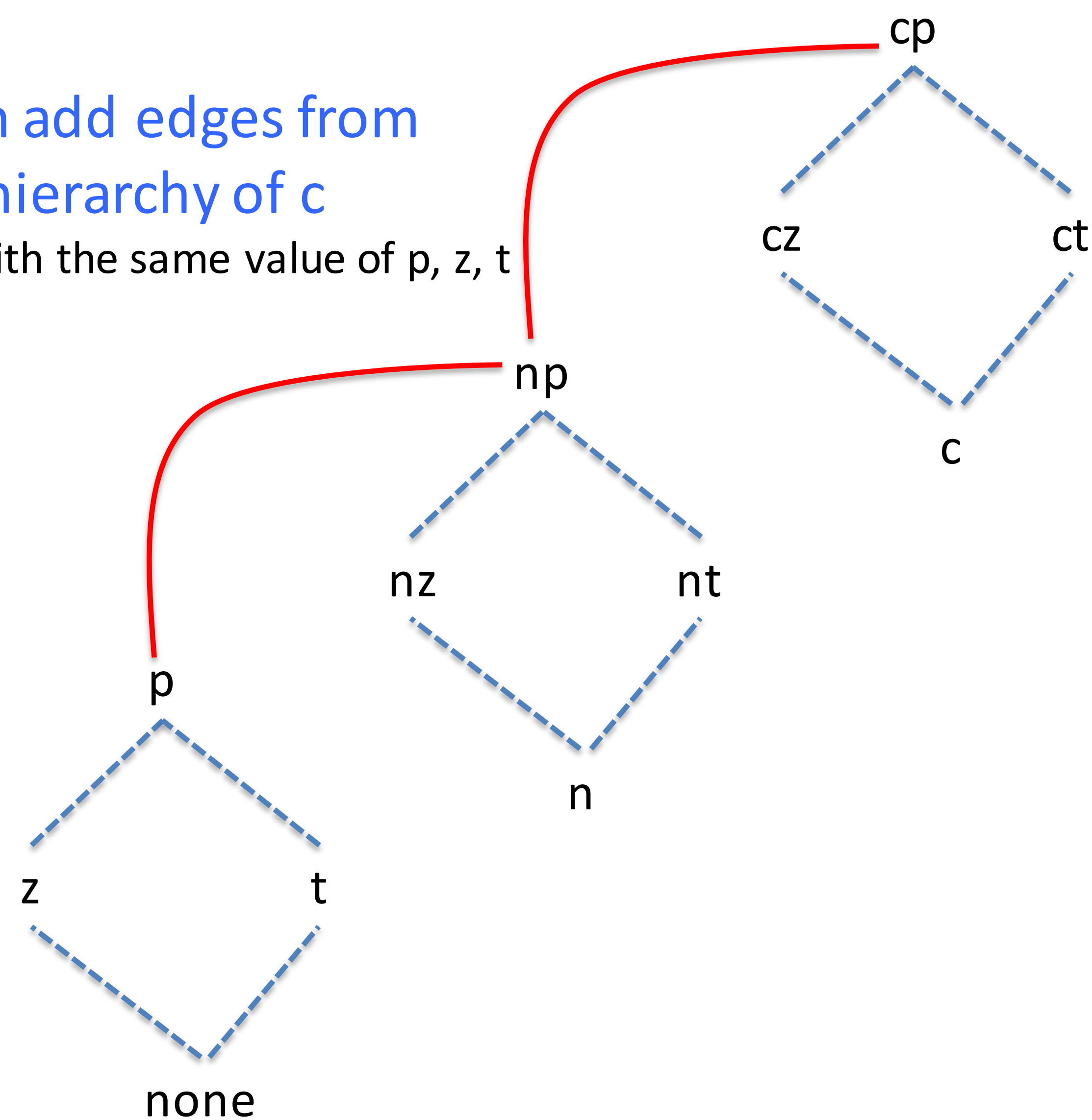


customer(c):
nation(n)



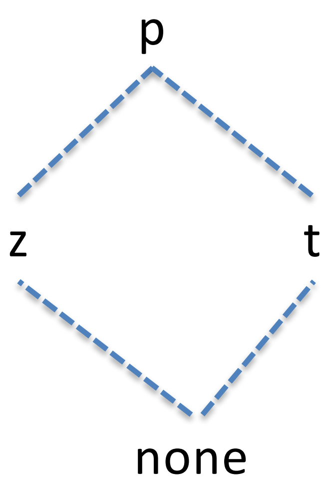
Then add edges from the hierarchy of c

- With the same value of p, z, t

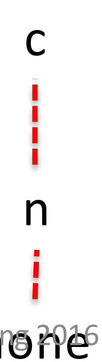


Combining Two Hierarchical Dimensions

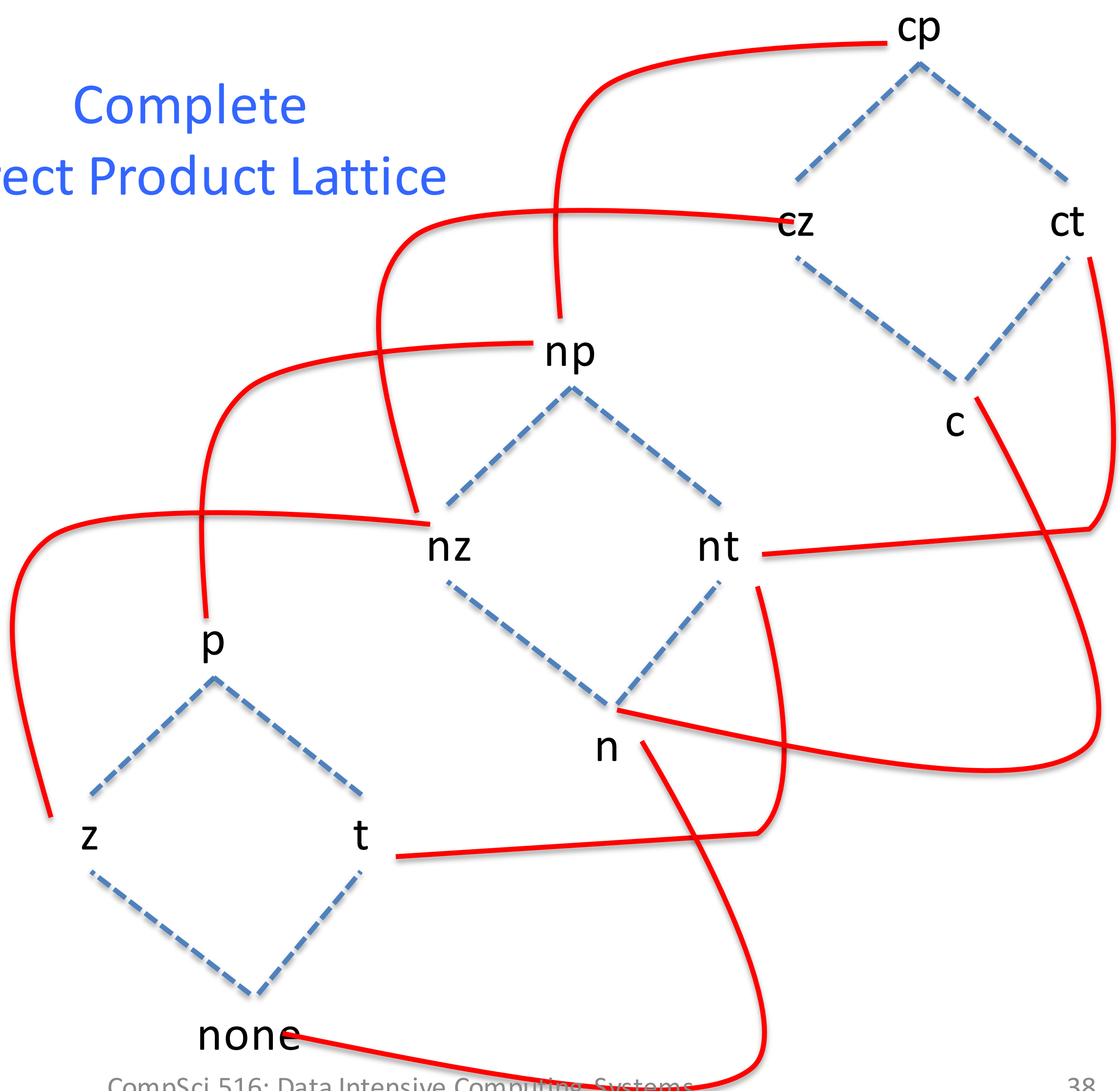
part:
size(z), type(t)



customer:
nation(n)



Complete
Direct Product Lattice



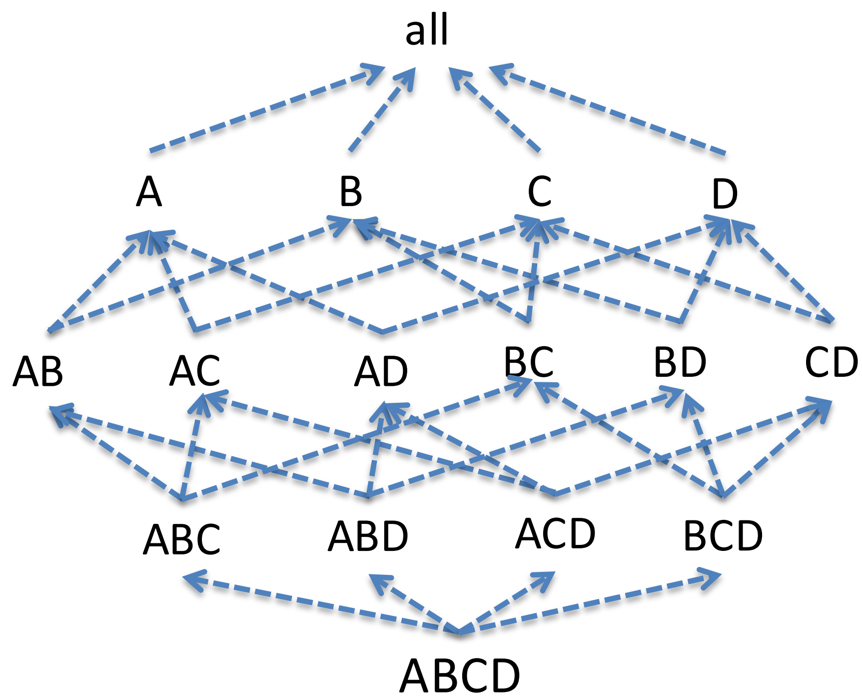
Implementing Data Cube

Basic Ideas

- Compute GROUP-BYs from previously computed GROUP-BYs
  - e.g. ABCD to (ABC or ACD) to (AB or AC) …
- Which order ABCD is sorted, matters for subsequent computations
  - if (ABCD) is the sorted order, ABC is cheap, ACD or BCD is expensive
- Next, some generic optimizations

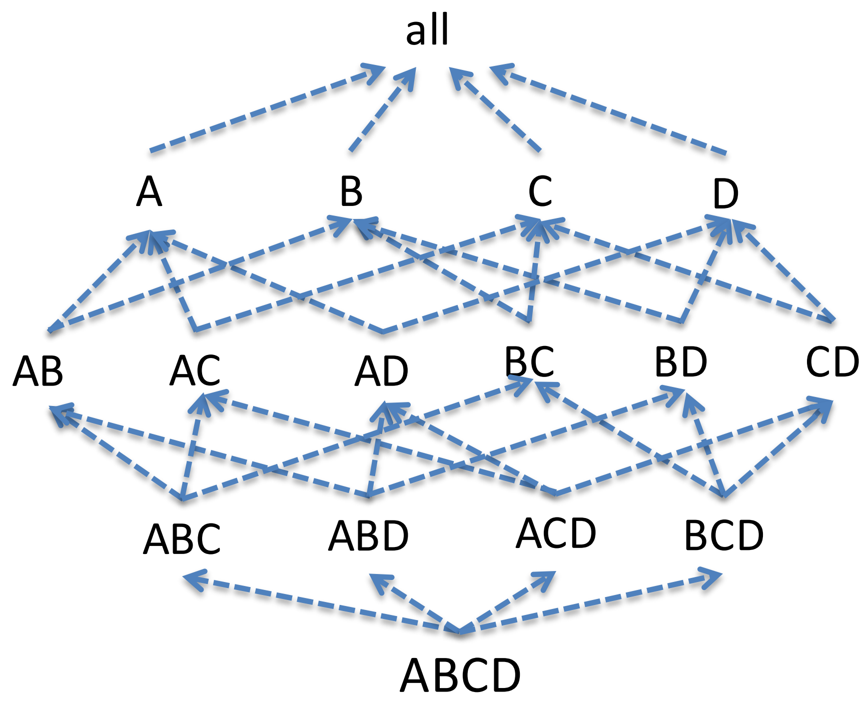
Optimization 1: Smallest Parent

- Compute GROUP-BY from the smallest (size) previously computed GROUP-BY as a parent
  - AB can be computed from ABC, ABD, or ABCD
  - ABC or ABD better than ABCD
  - Even ABC or ABD may have different sizes, try to choose the smaller parent



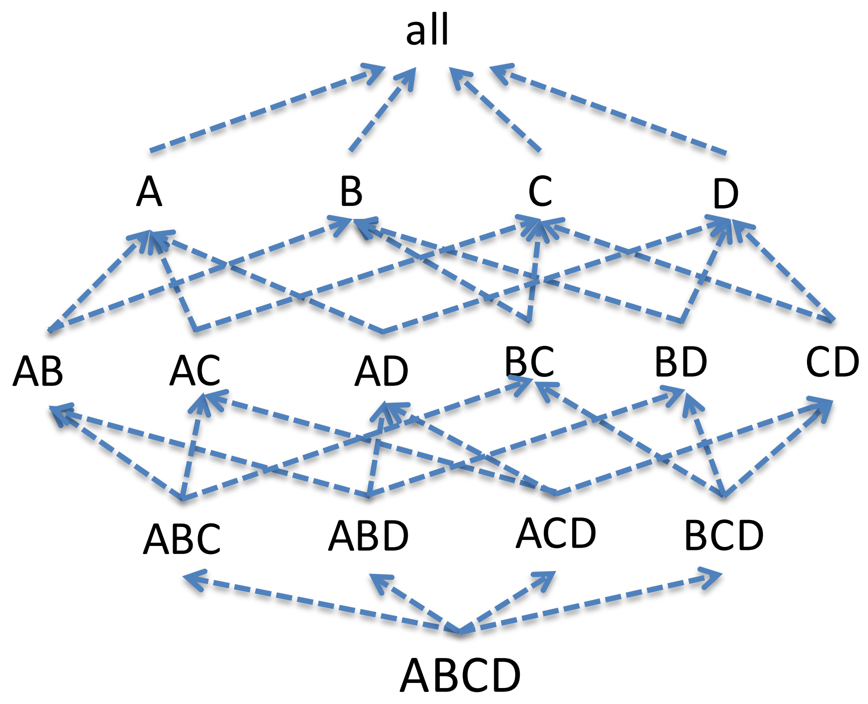
Optimization 2: Cache Results

- Cache result of one GROUP-BY in memory to reduce disk I/O
  - Compute AB from ABC while ABC is still in memory



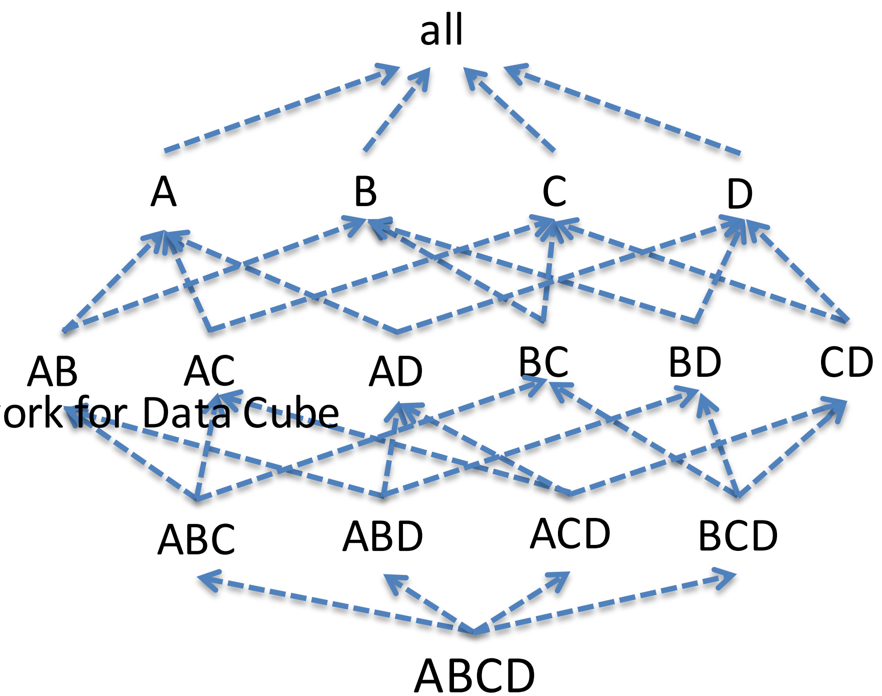
Optimization 3: Amortize Disk Scans

- Amortize disk reads for multiple GROUP-BYs
  - Suppose the result for ABCD is stored on disk
  - Compute all of ABC, ABD, ACD, BCD simultaneously in one scan of ABCD



Optimization 4, 5 (next)

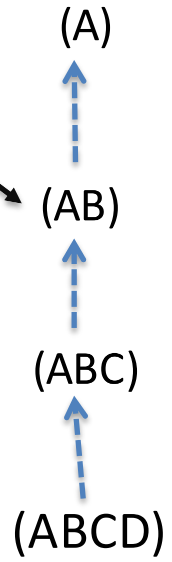
- 4. Share-sort
  - for sort-based algorithms
  - pipe-sort algorithm
  - covered in class
- 5. Shared-partition
  - for hash-based algorithms
  - pipe-hash algorithm
  - not covered - optional slides at the end



"Lattice" Framework for Data Cube

PipeSort: Idea

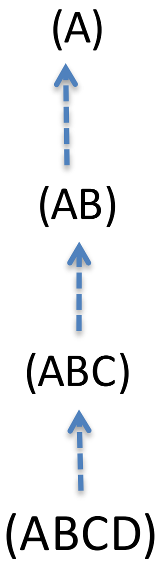
- Inside parenthesis (…): tuples sorted in this order
- No parenthesis: order can be arbitrary



- Combine two optimizations: "shared-sorts" and "smallest-parent"
- Also include "cache-results" and "amortized-scans"
  – Compute one tuple of ABCD, propagate upward in the pipeline by a single scan

PipeSort: Share-sort optimization

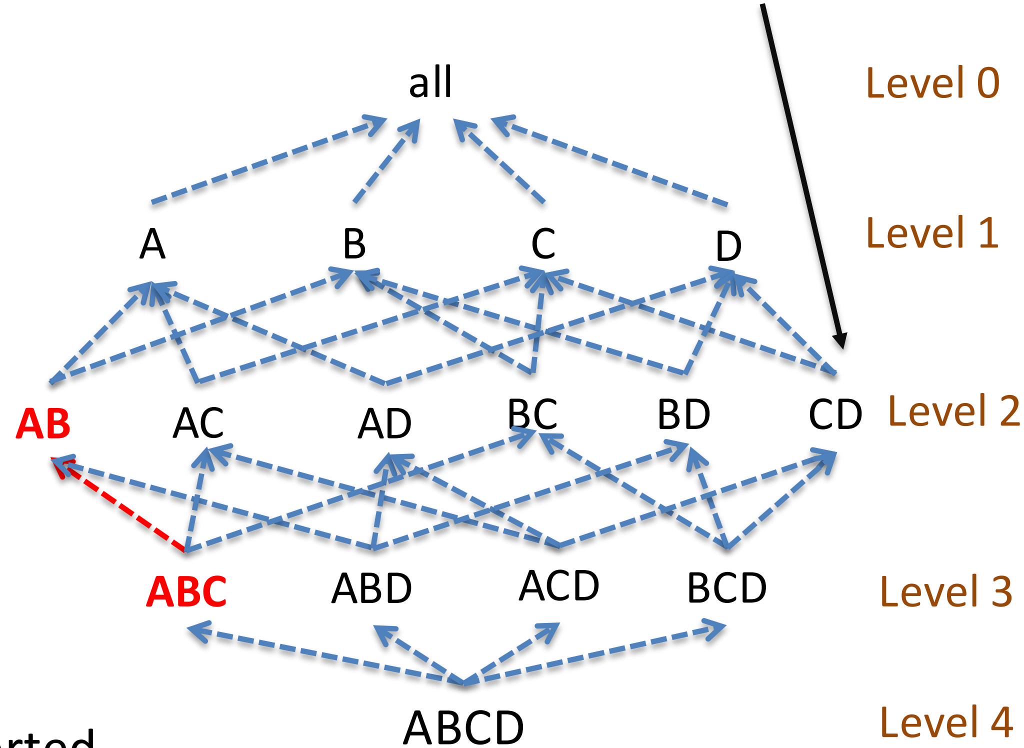
- Data sorted in one order
- Compute all GROUP-BYs prefixed in that order
- Example:
  - GROUP-BY over attributes ABCD
  - Sort raw data by (ABCD)
  - Compute (ABCD) -> (ABC) -> (AB) -> (A) in pipelined fashion
- No additional sort needed
- BUT, may have a conflict with "smallest-parent" optimization
  - ABD -> AB could be a better choice
  - Figure out the best parent choice by running a weighted-matching algorithm layer by layer



Search Lattice

- No parenthesis: order of tuples can be arbitrary

- Directed edge => one attribute less and possible computation
- Level k contains k attributes
  - all = 0 attribute
- Two possible costs for each edge e_{ij} = i ---> j
- **A(e_{ij}):** i is sorted for j
- **S(e_{ij}):** i is NOT sorted for j



Sorted

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a1	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

Not Sorted

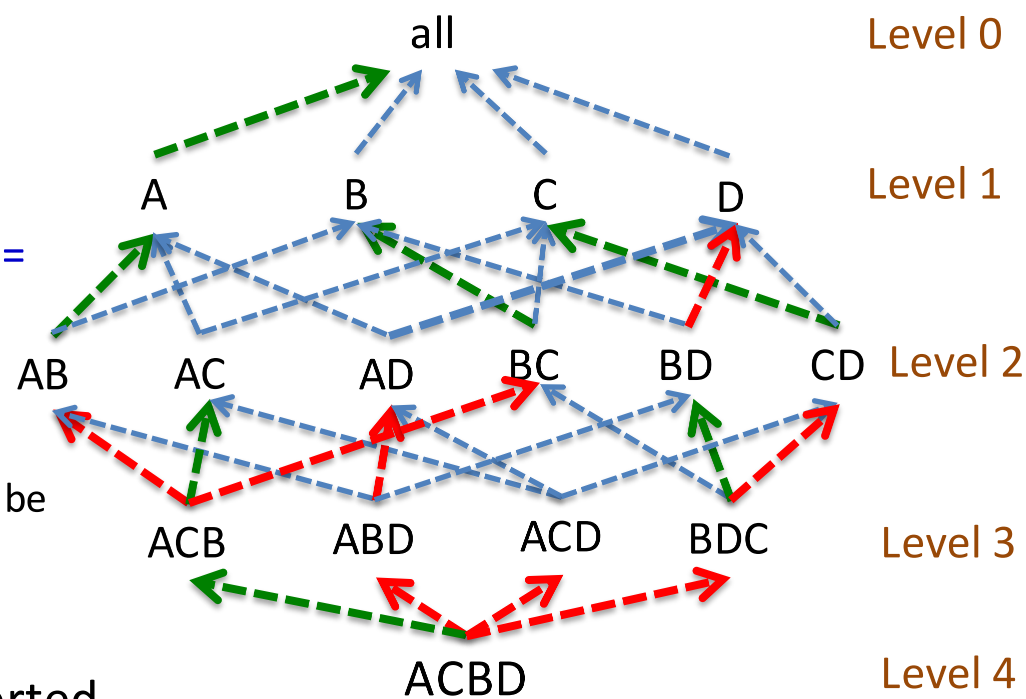
A	B	C	sum
a2	b2	c3	11
a1	b1	c2	10
a2	b2	c1	2
a1	b1	c1	5
a1	b2	c3	8

A	B	sum
a1	b1	15
a1	b2	8
a2	b2	13

Sorted (A)
Not-Sorted (S)

PipeSort Output

- A subgraph O
  - each node has a single parent
  - each node has a sorted order of attributes
- if parent's sorted order is a prefix, cost = **A(e_{ij})**, else **S(e_{ij})**
  - Mark by A or S
  - At most one **A-out-edge**
  - Note: for some nodes, there may be no **green A-out-edge**
- Goal: Find O with min total cost



Sorted

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a1	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

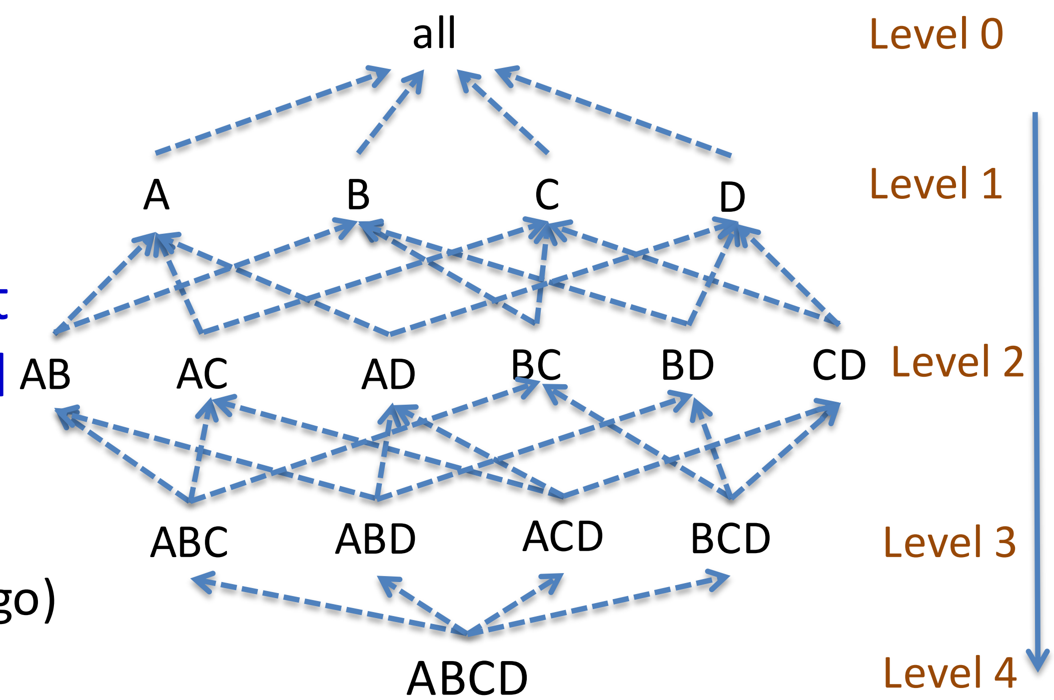
Not Sorted

A	B	C	sum
a2	b2	c3	11
a1	b1	c2	10
a2	b2	c1	2
a1	b1	c1	5
a1	b2	c3	8

A	B	sum
a1	b1	15
a1	b2	8
a2	b2	13

Outline: PipeSort Algorithm (1)

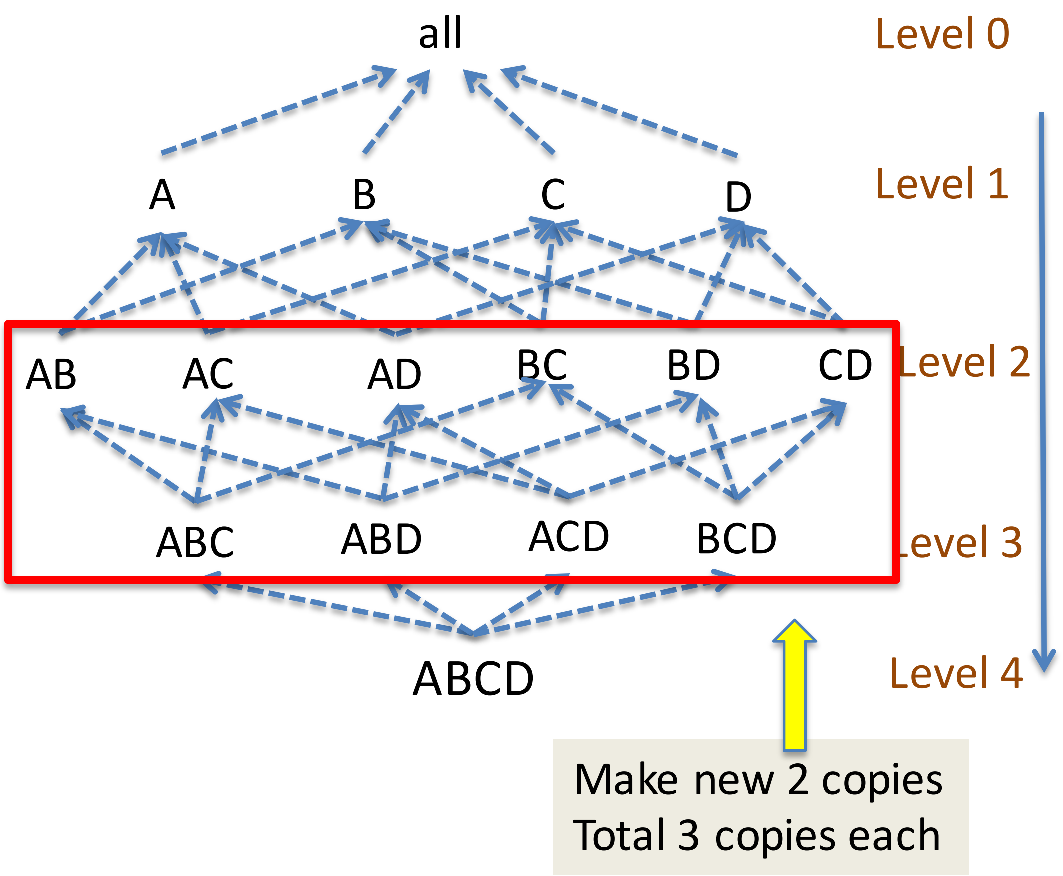
- Go from level 0 to N-1
  - here N = 4
- For each level k, find the best way to construct it from level k+1
  - use “min-cost weighted bipartite matching” (known algo)
    - Bipartite graph
    - vertices U, V
    - edges E with cost
    - choose a set of edges with min cost from E such that each vertex is matched with at most one vertex



Here V is large enough so that every vertex in U has a match (a parent node)

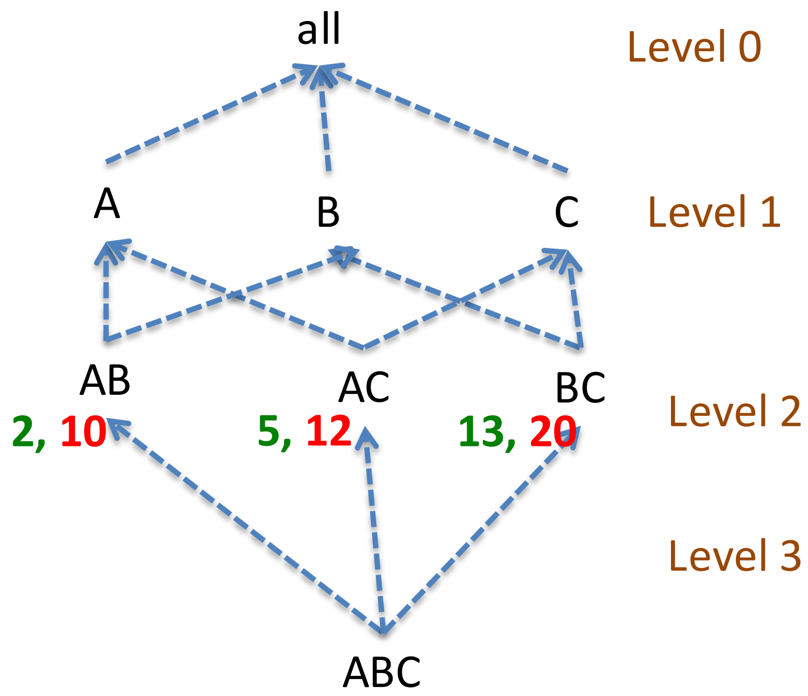
Outline: PipeSort Algorithm (2)

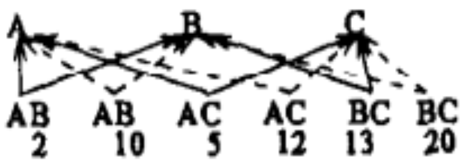
- A weighted bipartite matching between level k and k+1
- Make k new copies of each node in level k+1
  - k+1 copies for each in total
  - replicate edges
- Original copy = cost $A(e_{ij})$ = sorted
  - sorted order of i fixed according to j
- New copies = cost $S(e_{ij})$ = not sorted
  - need to sort i for j



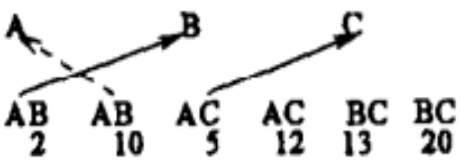
Outline: PipeSort Algorithm (3)

- Illustration with a smaller example
- Level k = 1 from level k+1 = 2
  - one new copy (dotted edges)
  - one existing copy (solid edge)
- Assumption for simplicity
  - same cost for all outgoing edges
  - $A(e_{ij}) = A(e_{ij'})$
  - $S(e_{ij}) = S(e_{ij'})$





(a) Transformed search lattice



(b) Minimum cost matching

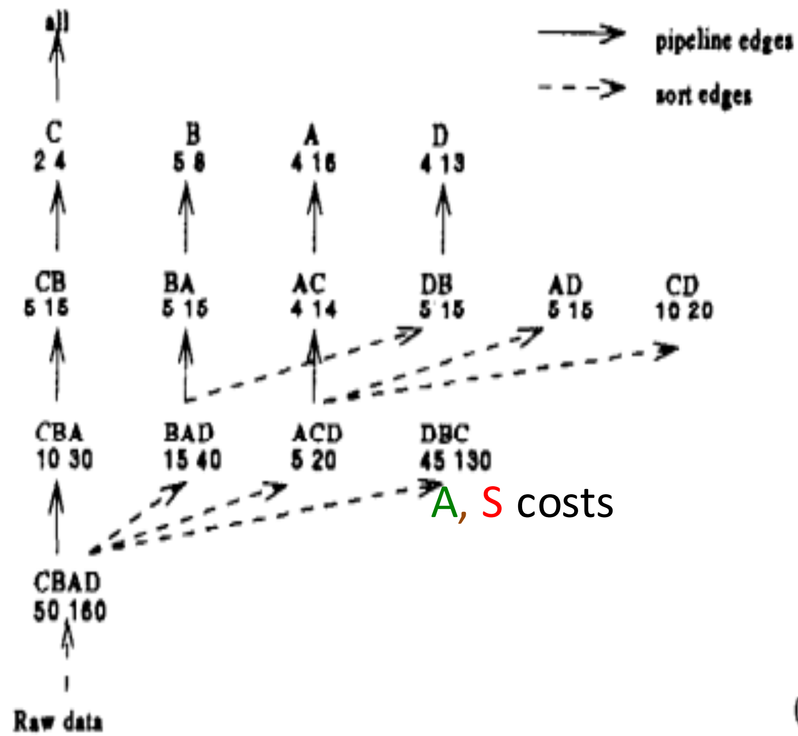
Outline: PipeSort Algorithm (4)

After computing the plan, execute all pipelines

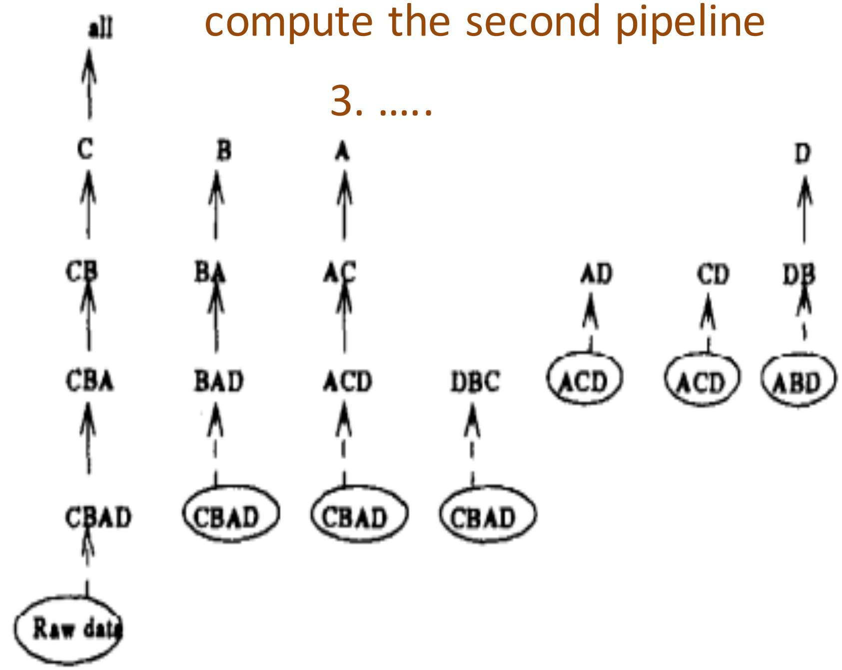
1. First pipeline is executed by one scan of the data

2. Sort (CBAD) -> (BADC),
compute the second pipeline

3. …..



(a) The minimum cost sort plan

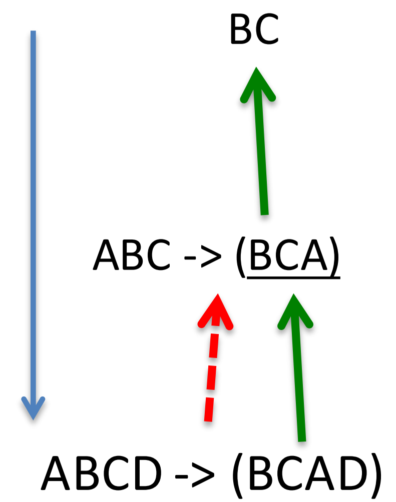


(b) The pipelines that are executed

Outline: PipeSort Algorithm (5)

Observations:

- Finds the best plan for computing level k from level k+1
  - Assuming the cost of sorting "BAD" does not depend on how the GROUP-BY on "BAD" has been computed
  - Generating plan k+1 -> k does not prevent generating plan k+2 -> k+1 from finding the best choice
- However, a heuristic and not provably globally optimal solution



If the green edge is chosen, the sorted order of ABCD will be BCAD

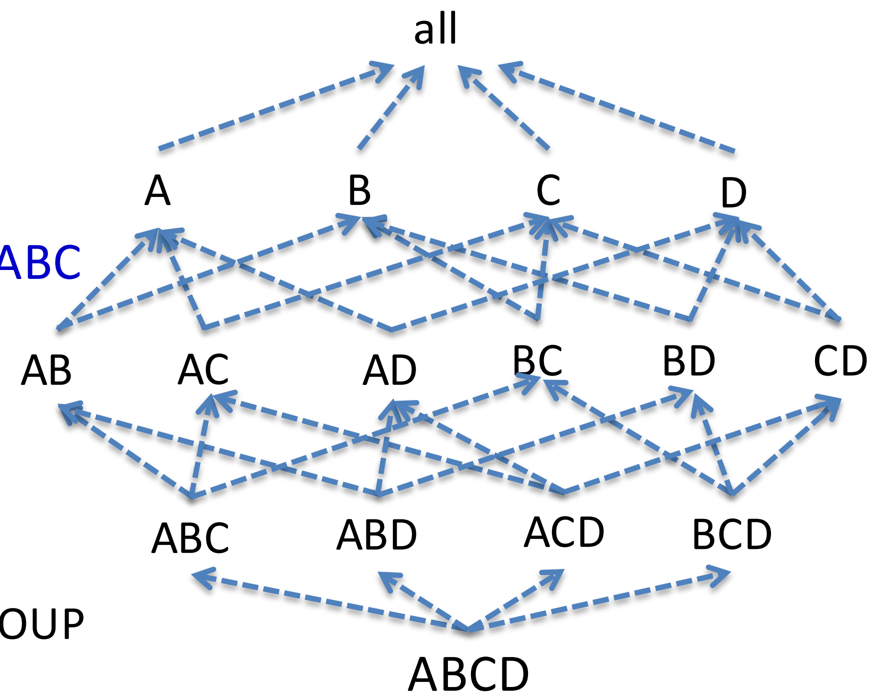
(Optional – additional slides)
PipeHash Algorithm

Optional material

PipeHash: Basic Idea (1)

N = 4

- Use hash tables to compute smaller GROUP-BYs
- If the hash tables for AB and AC fit in memory, compute both in one scan of ABC
- With no memory restrictions



for k = N...0:

For each k+1-attribute GROUP BY g

Compute in one scan of g all k-attribute GROUP BY where g is smallest parent

Save g to disk and destroy the hash table of g

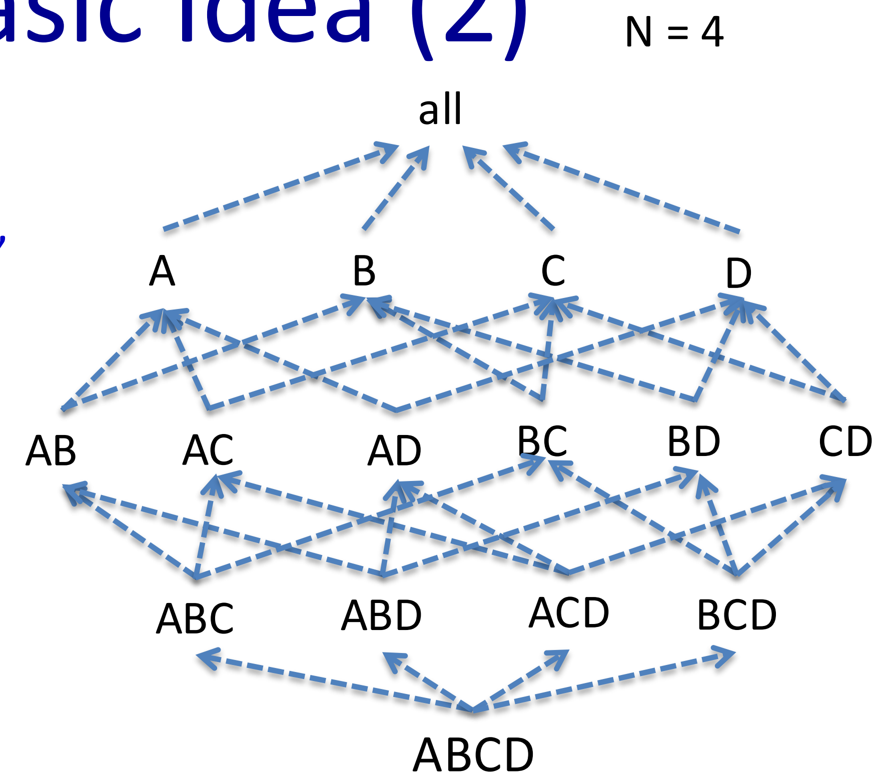
A	B		sum
a1	b1	→	15
a1	b2	→	8
a2	b2	→	13

A	C		sum
a1	c1	→	5
a1	c2	→	10
a2	c3	→	19
a2	c1	→	2

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a2	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

Optional material

PipeHash: Basic Idea (2)



- But, data might be large, Hash Tables may not fit in memory
- Solution: optimization "shared-partition"
  - partition data on one or more attributes
  - Suppose the data is partitioned on attribute A
  - All GROUP-Bys containing A (AB, AC, AD, ABC…) can be computed independently on each partition
  - Cost of partitioning is shared by multiple GROUP-BYs

A	B		sum
a1	b1	→	15
a1	b2	→	8
a2	b2	→	13

A	C		sum
a1	c1	→	5
a1	c2	→	10
a2	c3	→	19
a2	c1	→	2

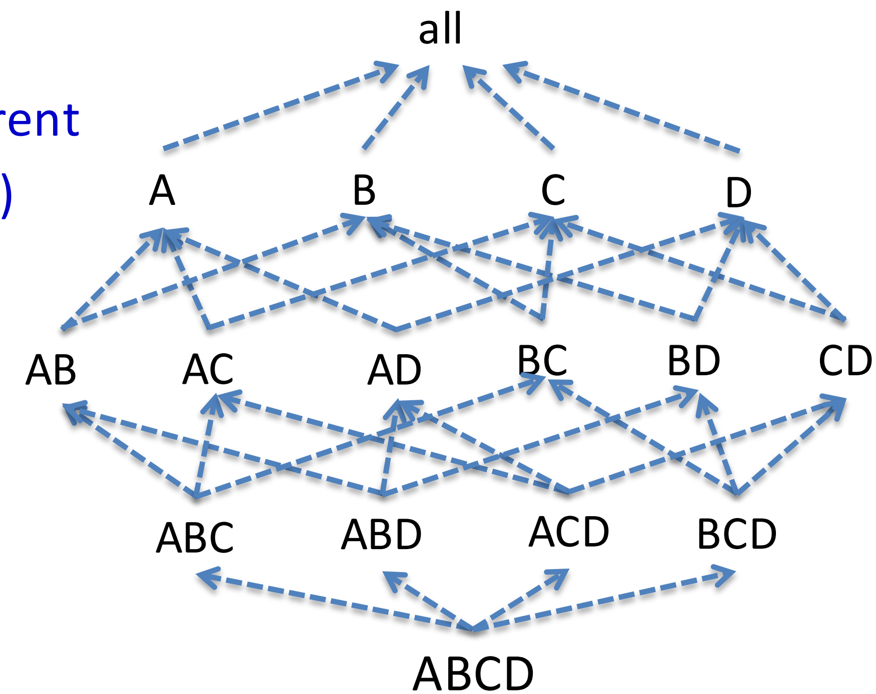
A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a2	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

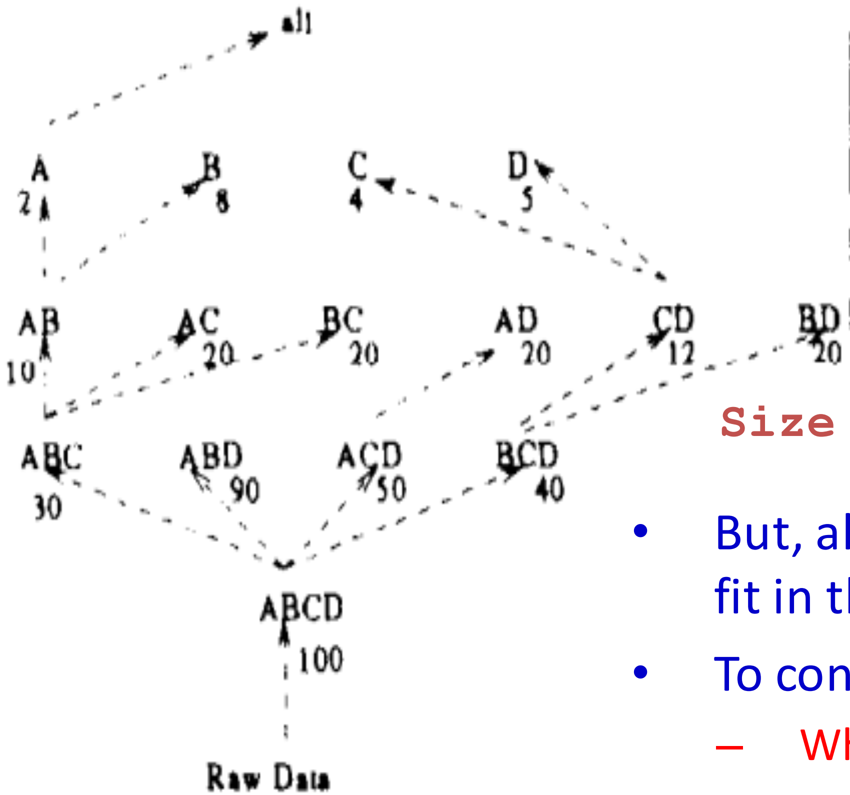
Optional material

PipeHash: Basic Idea (3)

N = 4

- Input: search lattice
- For each group-by, select smallest parent
- Result: Minimum Spanning Tree (MST)





(a) Minimum spanning tree

Size of GROUP-BY

- But, all Hash Tables (HT) in the MST may not fit in the memory together
- To consider:
  - Which GROUP-BYs to compute together?
  - When to allocate-release memory for HT?
  - What attributes to partition on?

Optional material

Outline: PipeHash Algorithm (1)

- Once again, a combinatorial optimization problem
- This problem is conjectured to be NP-complete in the paper
  - something to explore!
- Use heuristics

Trade-offs

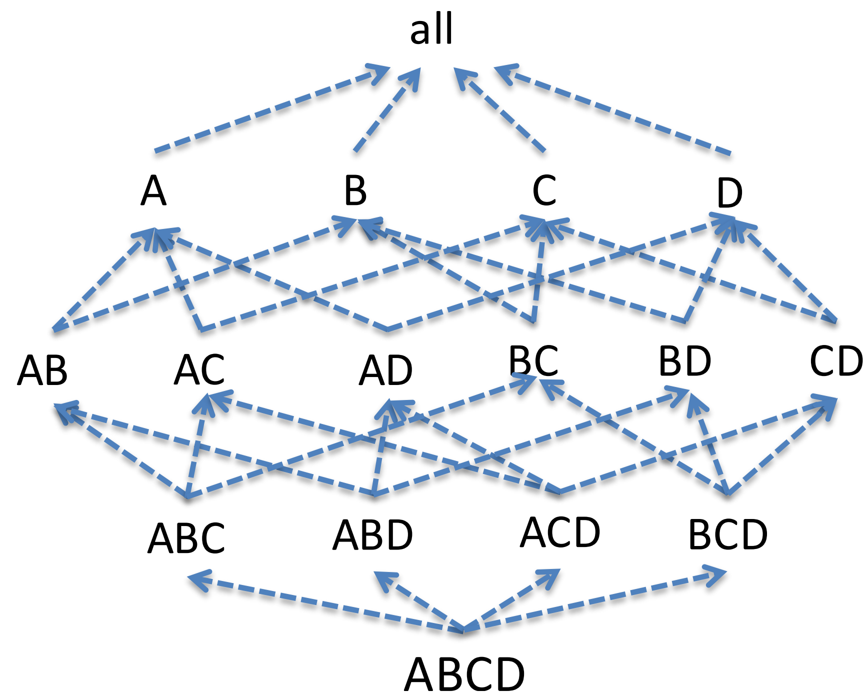
1. Choose as large sub-tree of MST as possible (“cache-results”, “amortized scan”)
2. The sub-tree must include the partitioning attribute(s)

Heuristic

Choose a partitioning attribute that allows selection of the largest subtree of MST

Optional material

Outline: PipeHash Algorithm (2)

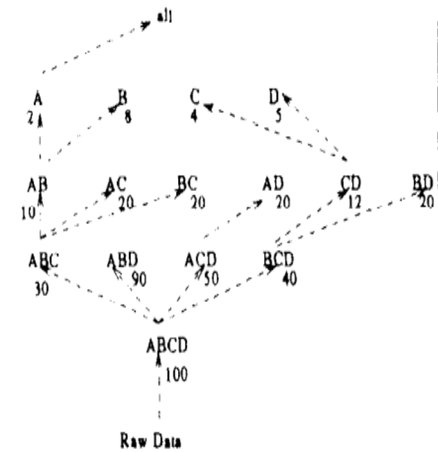


Algorithm

- Input: search lattice
- worklist = {MST}
- while worklist not empty
  - select one tree T from the worklist
  - T’ = select-subtree(T)
  - Compute-subtree(T’)

Next, through examples

- Select-subtree(T)
  - May add more subtrees to worklist
- Compute-subtree(T’)



(a) Minimum spanning tree

Optional material

Outline: PipeHash Algorithm (3)

- T' = Select-Subtree(T) = T_A
- Compute-Subtree(T')

Hash-Table in memory until all children are created

- s= {A} is such that
  - T_s per partition in P_s fits in memory

    P_s= #partitions
  - T' = T_s is the largest
- Creates new sub-trees to add

Partition T_A

For each partition,

Compute GROUP-BY ABCD

Scan ABCD to compute ABC, ABD, ACD

Save ABCD, ABD to disk

Compute AD from ACD

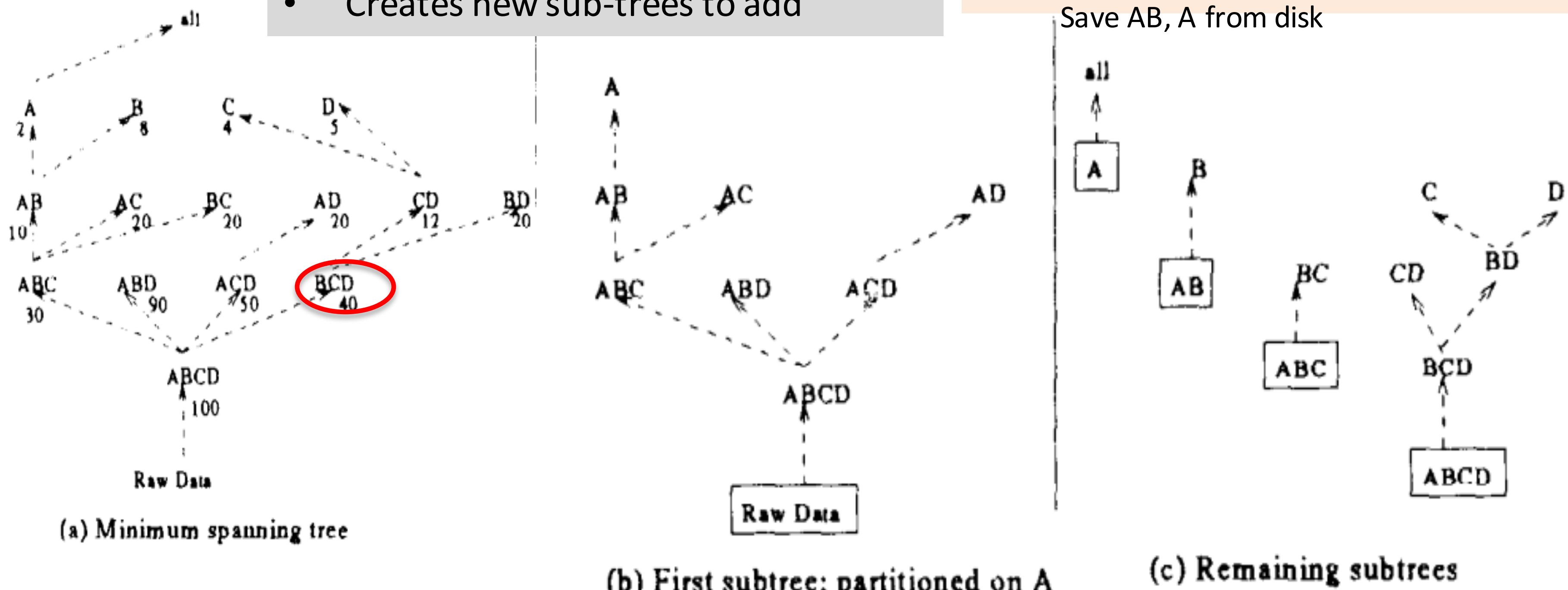
Save ACD, AD to disk

Compute AB, AC from ABC

Save ABC, AC to disk

Compute A from AB

Save AB, A from disk



(a) Minimum spanning tree

(b) First subtree: partitioned on A

(c) Remaining subtrees

Optional material

Experiments

5 Experimental evaluation

In this section, we present the performance of our cube algorithms on several real-life datasets and analyze the behavior of these algorithms on tunable synthetic datasets. These experiments were performed on a RS/6000 250 workstation running AIX 3.2.5. The workstation had a total physical memory of 256 MB. We used a buffer of size 32 MB. The datasets were stored as flat files on a local 2GB SCSI 3.5" drive with sequential throughput of about 1.5 MB/second.

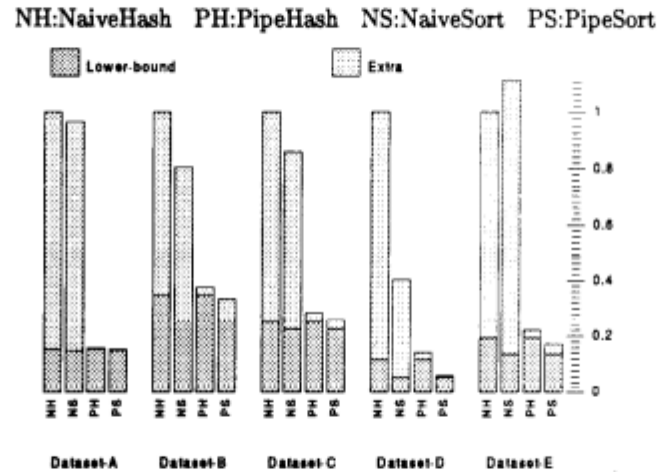


Figure 5: Performance of the cube computation algorithms on the five real life datasets. The y-axis denotes the total time normalized by the time taken by the NaiveHash algorithm for each dataset.

- Here sort-based better than hash-based (new hash-table for each GROUP-BY)
- Another experiment on synthetic data (see paper)
- For less sparse data, hash-based better than sort-based