Data Warehousing

- Decision support systems (DSS) in business
- On line analytical Processing (OLAP)
- Use data warehouses for analysis.
- complex queries on very large database, response time should be small

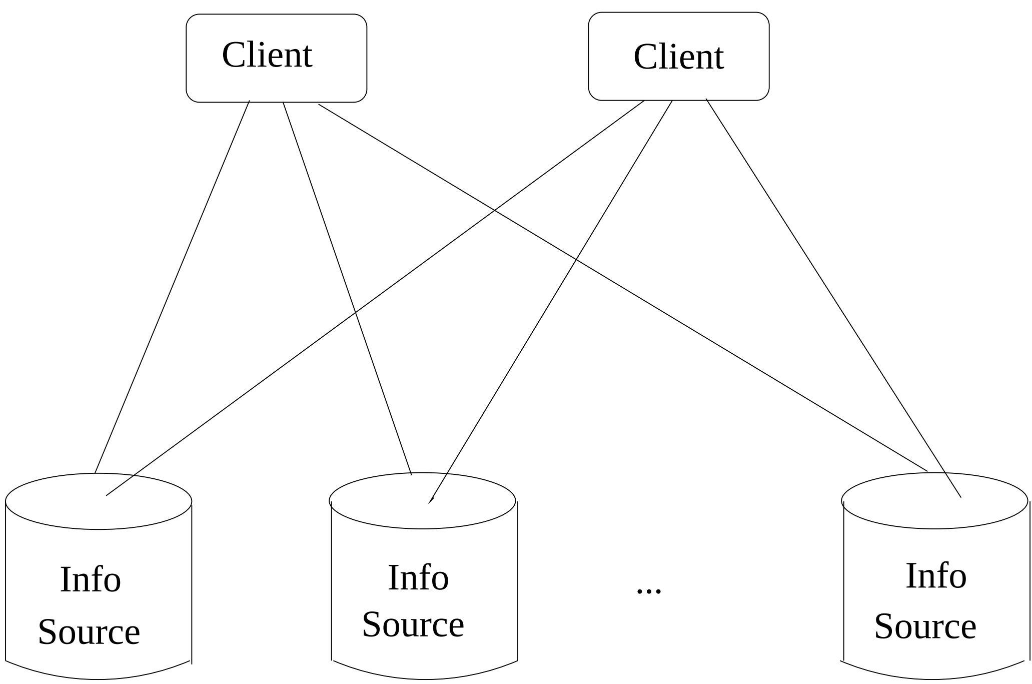
E.g.
R(Customer, District, Amount)

```
SELECT District, SUM(Amount)
FROM R
GROUP BY District,
```

- The raw data is relational data
- raw data $\rightarrow$ **data cube**
  with pre-computed answers
- Optimization: materialize some or all of these **cells** rather than compute them from raw data each time.
- Which cells to materialize when it is too expensive to materialize all views?

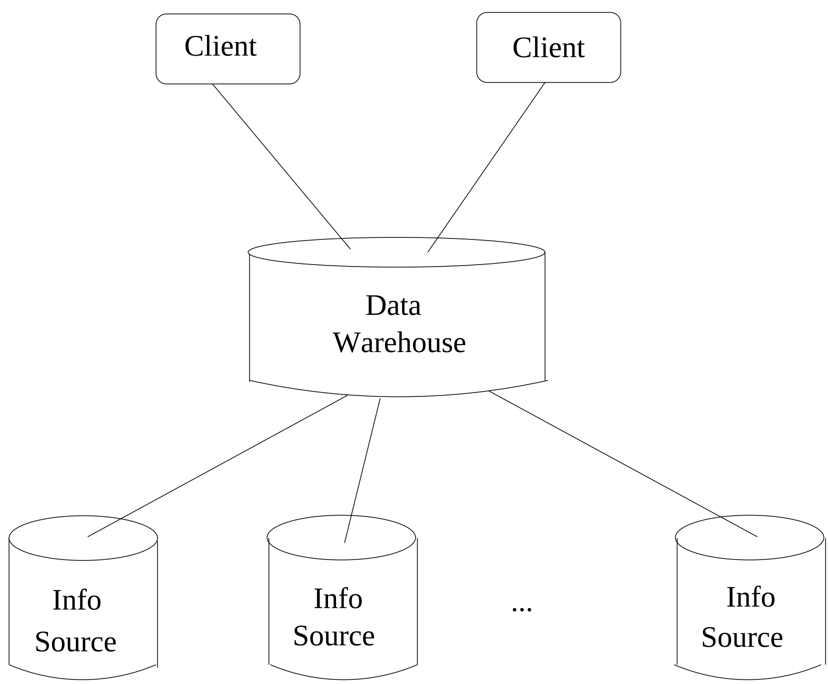
Heterogeneous database Integration

- Goal: easy and efficient access to integrated information from multiple, heterogeneous, autonomous, distributed information sources.



The Warehousing Approach

- Information integrated in advance
- Stored in warehouse for direct querying and analysis

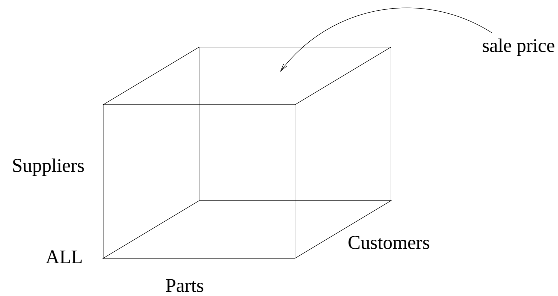


Advantages of Warehousing Approach

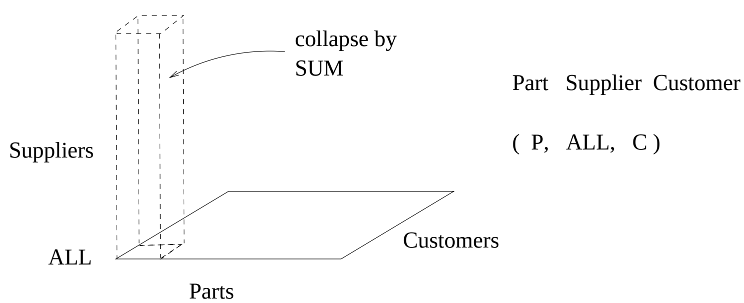
- High query performance
  But not necessarily most recent state
- Doesn't compete with local processing
- Information copied at warehouse
  - Can modify, annotate, summarize, etc.
  - Can store historical information

The Data Cube

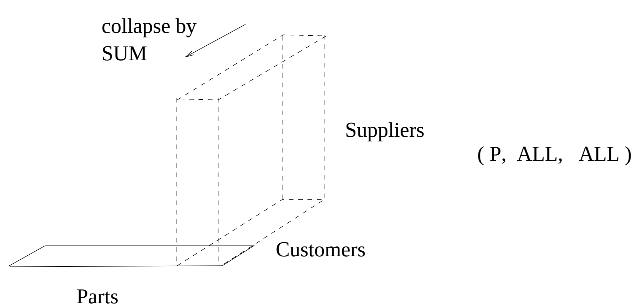
E.g. Parts are bought from suppliers and then sold to customers at a sale price SP

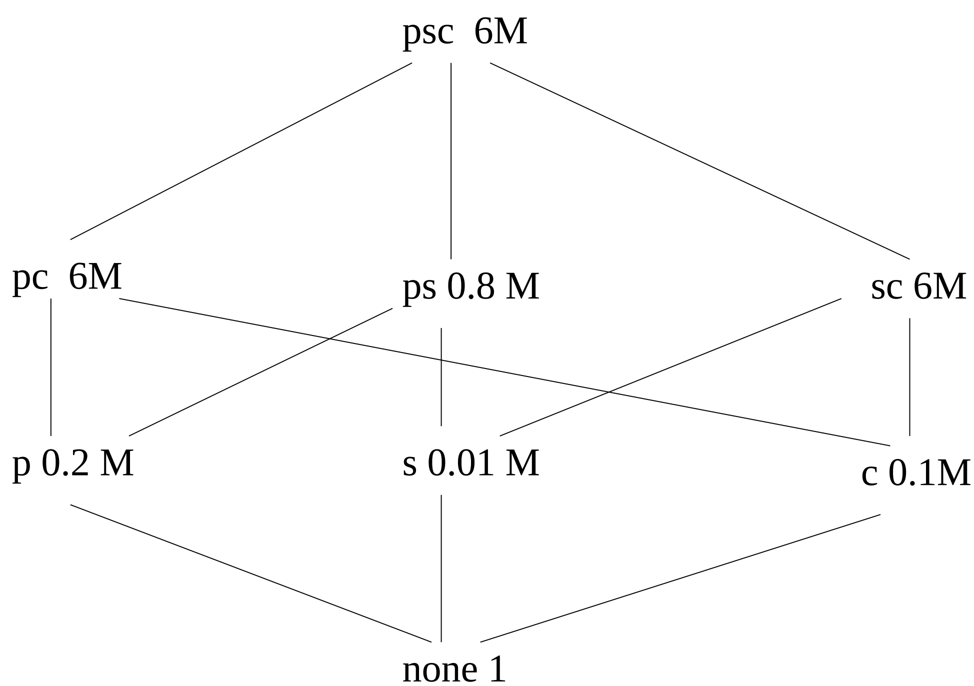


SELECT Part, Customer, SUM(SP)
FROM R
GROUP BY Part, Customer



SELECT Part, SUM(SP)
FROM R
GROUP BY Part

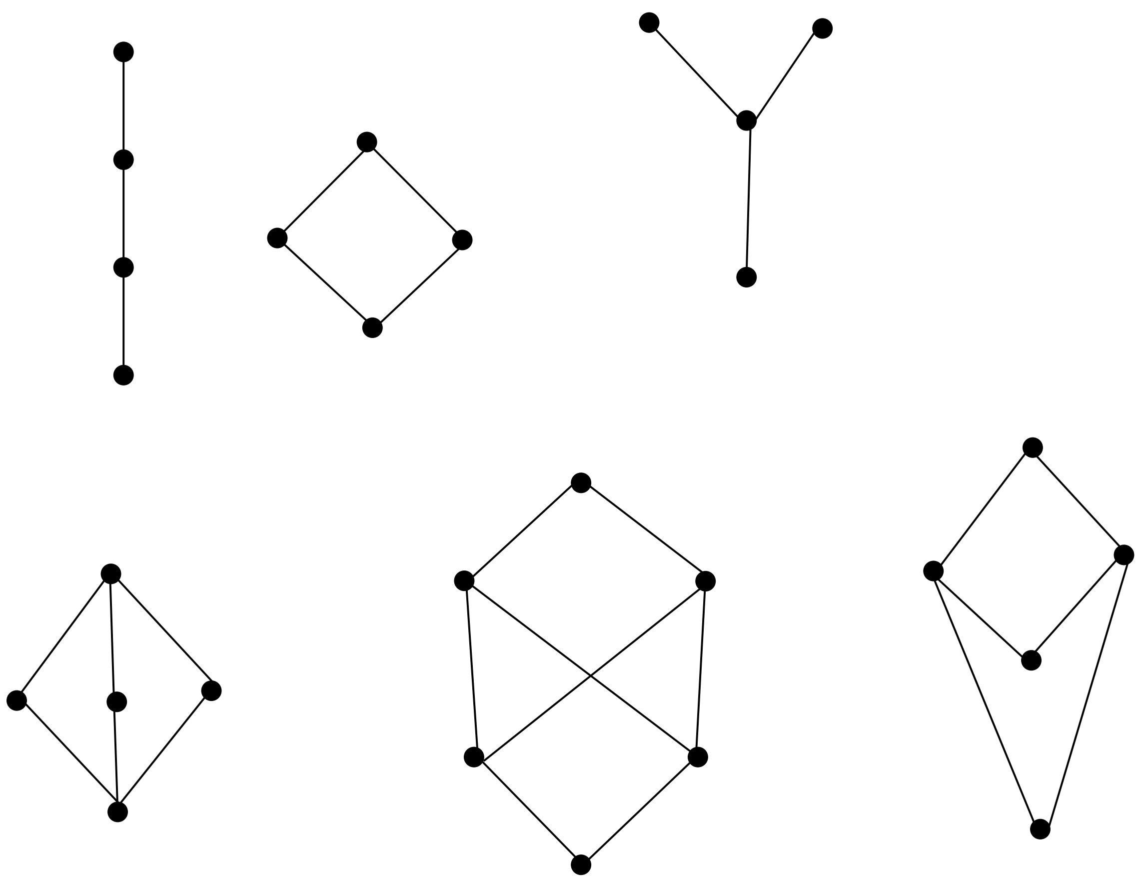




1. part, supplier, customer (6M, i.e., 6 million rows)
2. part, customer (6M)
3. part, supplier (0.8M)
4. supplier, customer (6M)
5. part (0.2M)
6. supplier (0.01M)
7. customer (0.1M)
8. none (1)

The Lattice Framework

A lattice is a poset $(L, \leq)$ in which every subset $\{a, b\}$ consisting of two elements has a least upper bound a unique greatest lower bound.



The Lattice Framework

- Consider 2 queries Q_1, Q_2

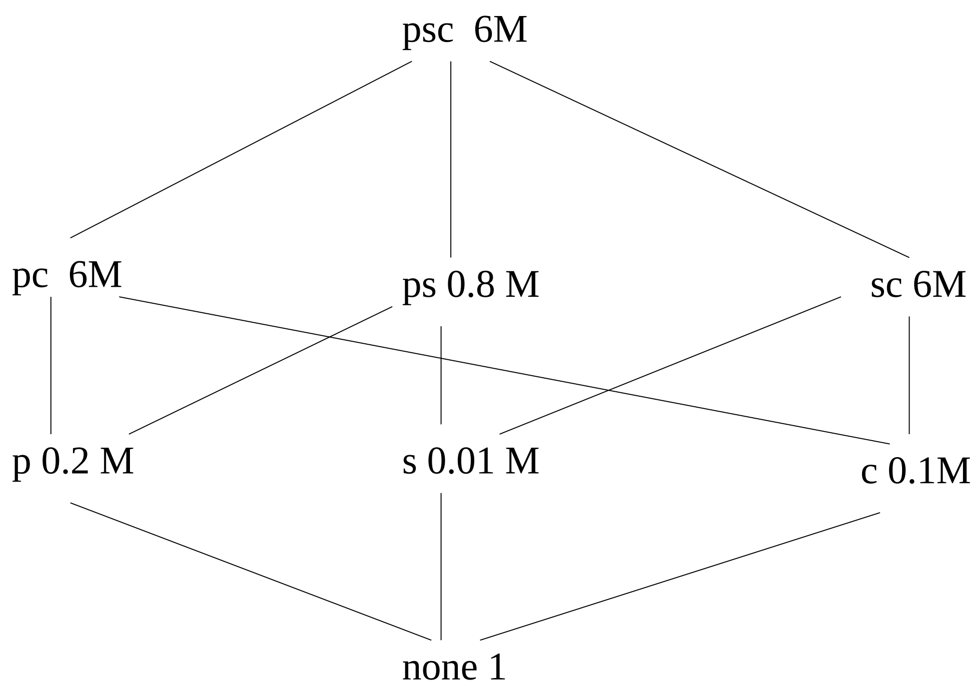
  We say $Q_1 \leq Q_2$ if and only if

  Q_1 can be answered using only the results of Q_2.

  Q_1 is **dependent** on Q_2

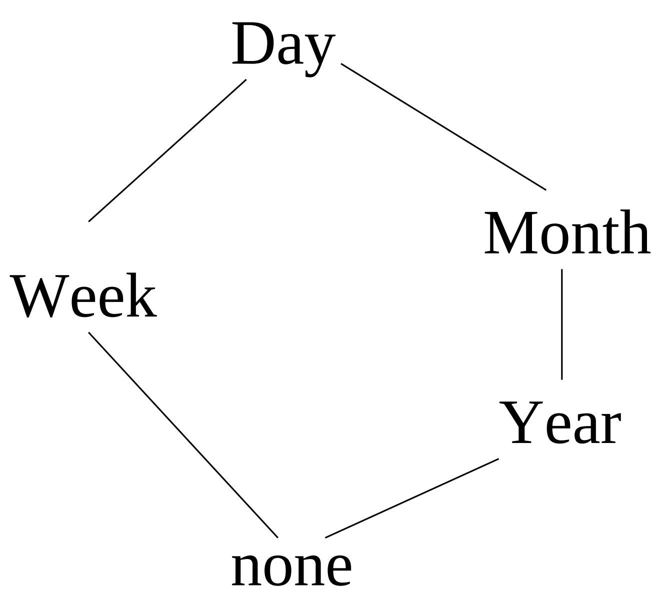
  e.g. (part) $\leq$ (part, customer)

  (part) $\not\leq$ (customer)

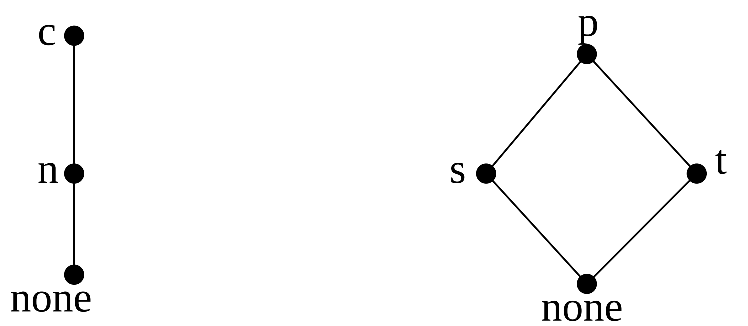


The Lattice Framework (cont.)

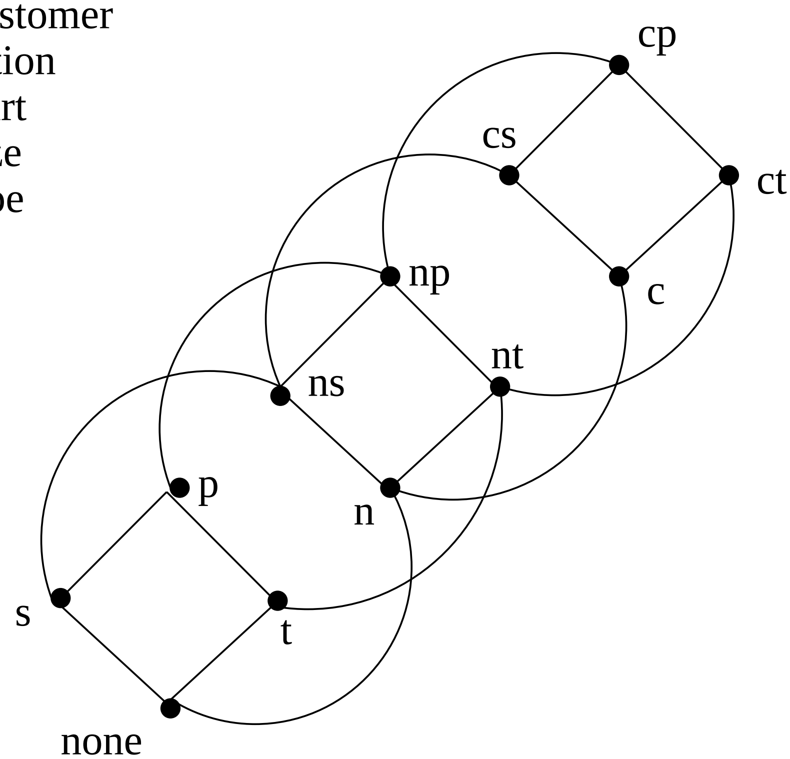
- Also, hierarchy such as day, month year.
  common querying operations:
  drill-down : per year $\longrightarrow$ per month
  roll-up : per day $\longrightarrow$ month $\longrightarrow$ year



Combining lattices



c - customer
n- nation
p - part
s - size
t - type



Theorem: If L_1, L_2 are lattices, then $L_1 \times L_2$ is a lattice.

$\boxed{\text{MOLAP vs ROLAP}}$

Two ways of storing the datacube for OLAP:

MOLAP - Multidimensional OLAP

Aggregates are stored in a multi-dimensional cube, which is not in the relational database.

E.g. Product $\times$ Company $\times$ Month

(product-x, A, Jan): $204,000
(Product-x, B, Jan): $0
(Product-x, ALL, Jan): $204,000

(Product-x,B,Jan) is **empty**.

- Can facilitate fast retrieval, however, many entries may be empty

- E.g. 10 dimensions, each with 10 values, ALL resembles an extra value.

- $11^{10} - 10^{10} = 15{,}937{,}424{,}601$ counters are needed for the aggregate cells.

- Many aggregate cells are empty.

- $10^{10} = 10$ billions. Even if a base table has 1 billion tuples, 90% of the base cells and many aggregate cells are empty.

ROLAP – Relational OLAP

the data cube is stored in the relational database to support OLAP functions.

$\boxed{\text{Multiway Aggregate Computation}}$

Compute ABC

Compute AB, AC, BC

Compute A, B, and C

Compute ALL

Note that A, B, C can be computed from any two of AB, AC, BC

Different sorting order can affect the computation cost.

Suppose $|A| = 10, |B| = 100, |C| = 1000$

if ABC is sorted in the order of A-B-C:

$a_1b_1c_1, a_1b_1c_2, ..., a_1b_1c_{1000}$
$a_1b_2c_1, ...,\ a_1b_{100}c_1, ...$
$a_2b_1c_1, ..., a_2b_2c_1..,$
$\dots,$
$a_{10}b_1c_1, ...a_{10}b_2c_1, ...$

- To get (A,B,*):

  sum all $a_1b_1c_1, ...a_1b_1c_{1000}$
  sum all $a_1b_2c_1, ...a_1b_2c_{1000}$
  ... sum all $a_1b_{100}c_1, ...a_1b_{100}c_{1000}$
  ... sum all $a_{10}b_{100}c_1, ...a_{10}b_{100}c_{1000}$

  one scan

- To get (A,*,C):
  sum all $a_1b_1c_1$, $a_1b_2c_1$,... $a_1b_{100}c_1$

  1000 scans in total.

- To get (*,B,C):
  sum all $a_1b_1c_1$, $a_2b_1c_1$, ... $a_{10}b_1c_1$

  100×1000 scans.

If ordered in C-B-A:

$c_1b_1a_1, c_1b_1a_2, ..., c_1b_1a_{10}$
$c_1b_2, a_1, ...,\ c_1b_{100}a_1, ...$
$c_2b_1a_1, ..., c_2b_2a_1..,$
$\ldots,$
$c_{1000}b_1a_1, ...c_{1000}b_2a_1, ...$

- To get (A, B, *):

  sum all $a_1b_1c_1, a_1b_1c_2, ..., a_1b_1c_{1000}$

  10×100 scans

- (A,*,C): 10 scans
  (*,B,C): 1 scan
  (A,*,*): 10 scans
  (*,B,*): 100 scans

The C-B-A order allows much lower overall computation cost.

Indexing for OLAP

Bitmap Indexes

sailor-id	gender	rating
112	M	3
120	M	5
130	F	2
134	M	5

M	F
1	0
1	0
0	1
1	0

1	2	3	4	5
0	0	1	0	0
0	0	0	0	1
0	1	0	0	0
0	0	0	0	1

Advantages over hashing or tree indexing

1. Allow the use of efficient bit operations

   E.g. number of male customers with rating of 5:

   bitwise AND

   (bit for gender) AND (5th bit for rating)

   then count number of 1's in the resulting bit vector.

2. can be much more compact, allow the use of compression techniques.

$\boxed{\text{Implementing Data Cubes Efficiently}}$

V. Harinarayan, A. Rajaraman, J. D. Ullman
SIGMOD 1996

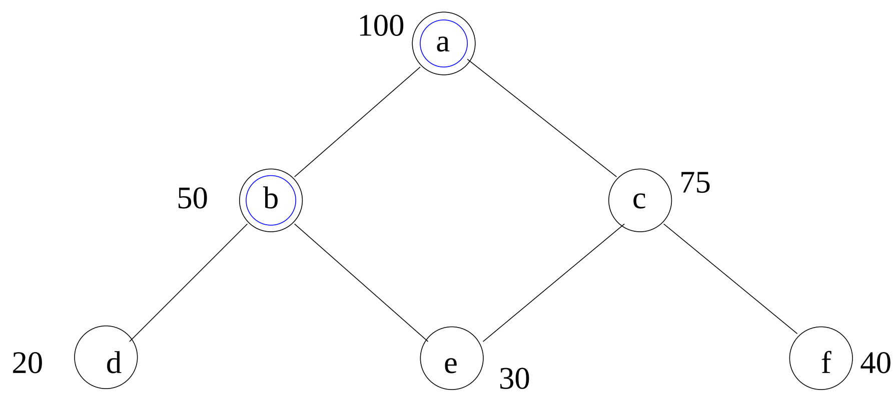
The Cost Model: Linear to size

- Query = some view in lattice

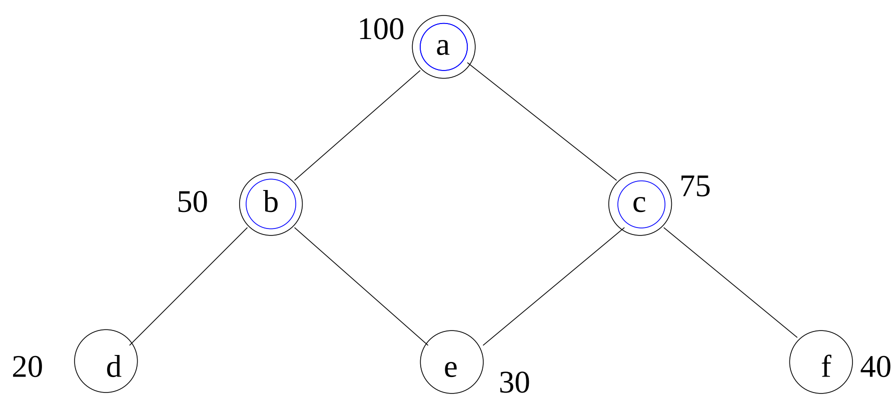
- Cost of answering Q = number of rows in the table for constructing the view

$$\boxed{T = m \times S + c}$$

where S is the view size



c: from a – cost 100
d: from b – cost 50
e: from b – cost 50
f: from a – cost 100



f: from c – cost 75
gain = 25
c: from c – cost 75
gain = 25

Benefit of c given a b, is 25 + 25

A simple optimization problem:

1. minimize the average time taken to evaluate a view
2. constrained to materialize a fixed number of views

Assume each view has the same chance of being requested.

—

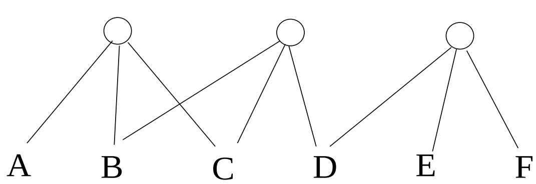
NP-complete

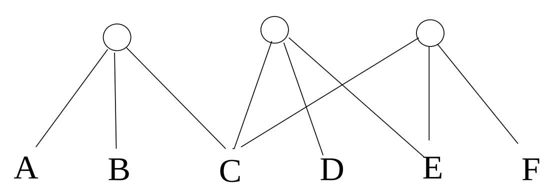
Exact cover by 3-sets

Instance: Set X with $3q$ elements a collection C of 3-elements subsets of X

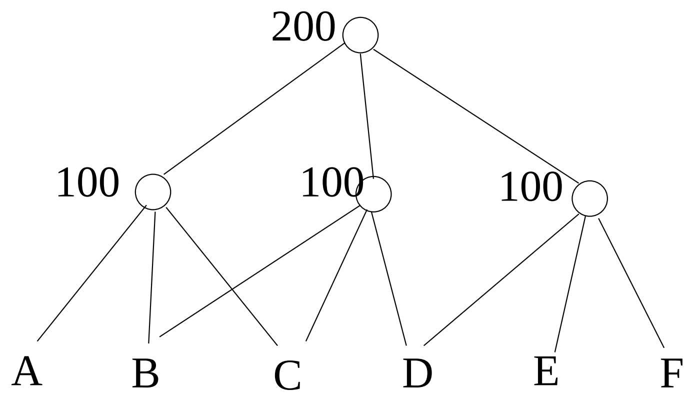
Question: Does C contain an exact cover for X,

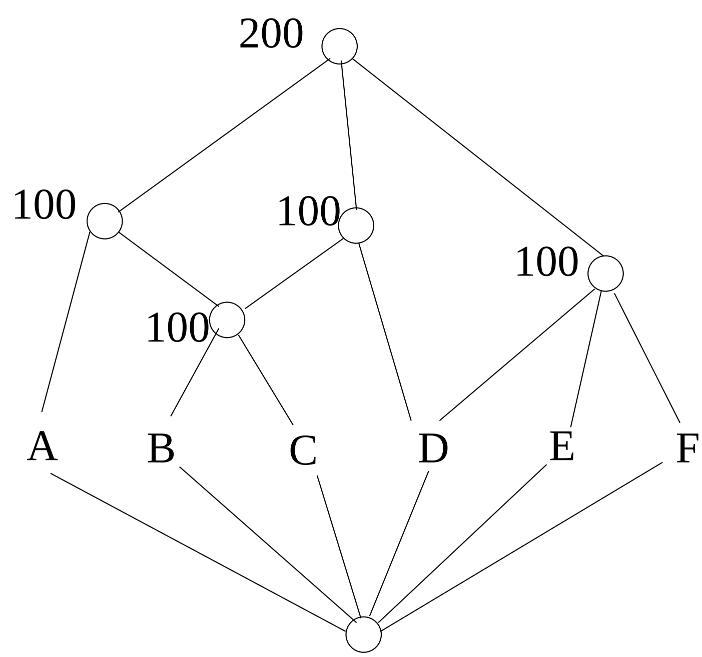
i.e. a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'?





$q = 2$
Create lattice for the first example:





Set k (number of views) = q = 2

The Greedy Algorithm

Let $C(v)$ be the cost of view v. = number of rows in view v.

Number of views to materialize $= k$

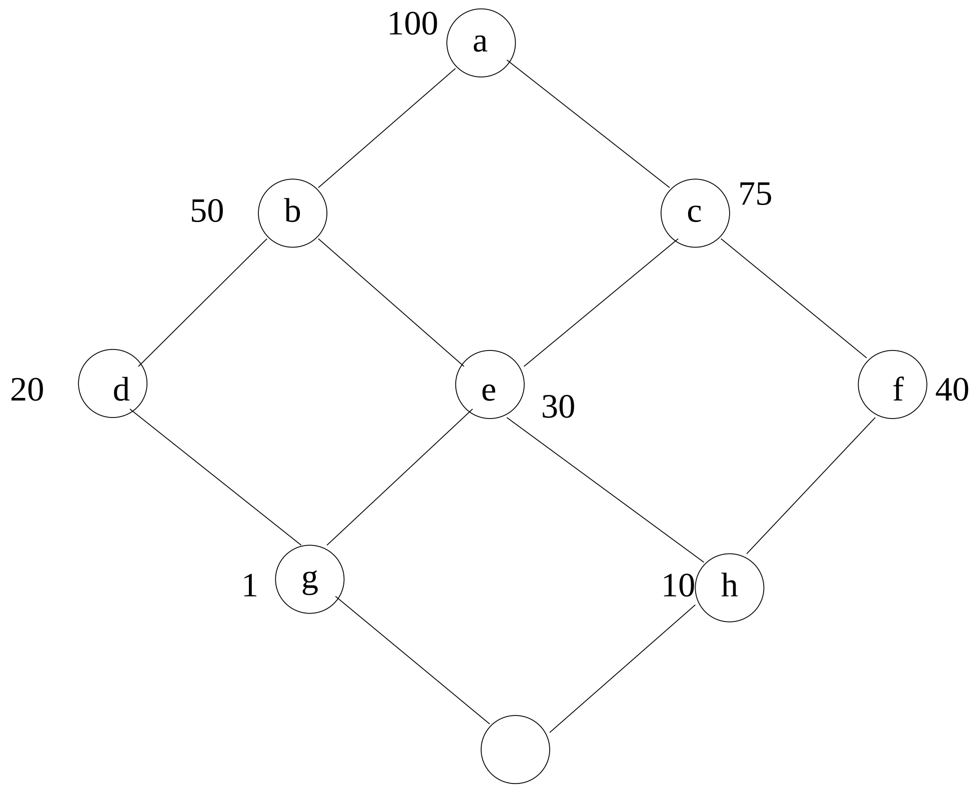
B(v,S): After selecting a set S of views, the benefit of view v.

1. For each $w \leq v$, define the quantity B_w by
   (a) Let u be the view of least cost in S such that $w \leq u$. *
   (b) If $C(v) < C(u)$, then $B_w = C(v) - C(u)$. Otherwise, $B_w = 0$.
2. Define $B(v, S) = \sum_{w \leq v} B_w$.

*Note that since the top view is in S, there must be at least one such view in S.

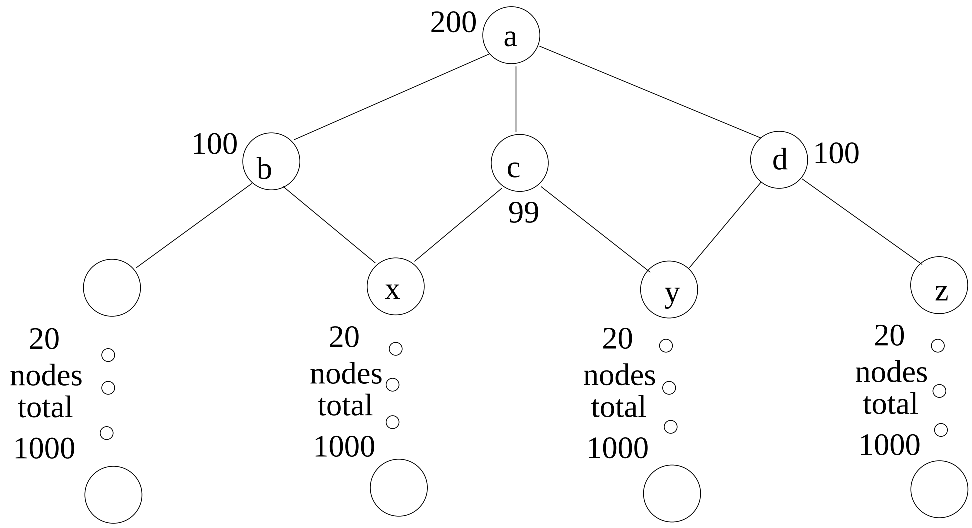
The Greedy Algorithm:

```
S = {top view};

for i = 1 to k do begin

    select that view v not in S such that
           B(v,S) is maximized;

    S = S union v;

end;

resulting S is the greedy selection;
```



	First Choice	Second Choice	Third Choice
b	$50 \times 5 = 250$		
c	$25 \times 5 = 125$	$25 \times 2 = 50$	$25 \times 1 = 25$
d	$80 \times 2 = 160$	$30 \times 2 = 60$	$30 \times 2 = 60$
e	$70 \times 3 = 210$	$20 \times 3 = 60$	$20 + 20 + 10 = 50$
f	$60 \times 2 = 120$	$60 + 10 = 70$	
g	$99 \times 1 = 99$	$49 \times 1 = 49$	$49 \times 1 = 49$
h	$90 \times 1 = 90$	$40 \times 1 = 40$	$30 \times 1 = 30$

Benefits of possible choices at each round



Let $k = 2$

Greedy: First pick c, benefit = 4141

(c and 40 views below it
each improve from 200 to 99
hence 101×41)

Second pick b, benefit = 2100

Optimal Choice:

b,d, each one's benefit = $100 \times 41 = 4100$

$$\frac{Greedy}{Optimal} = \frac{6241}{8200} = \frac{3}{4}$$

$\boxed{\text{A Performance Guarantee}}$

The greedy algorithm result benefit result is at least A/B of the optimal benefit

where

$$A/B = 1 - \left(\frac{k-1}{k}\right)^k$$

For example, when $k = 2$,

$$A/B = 1 - \left(\frac{2-1}{2}\right)^2 = \frac{3}{4}$$