Chapter 9: Rule Mining

9.1 OLAP

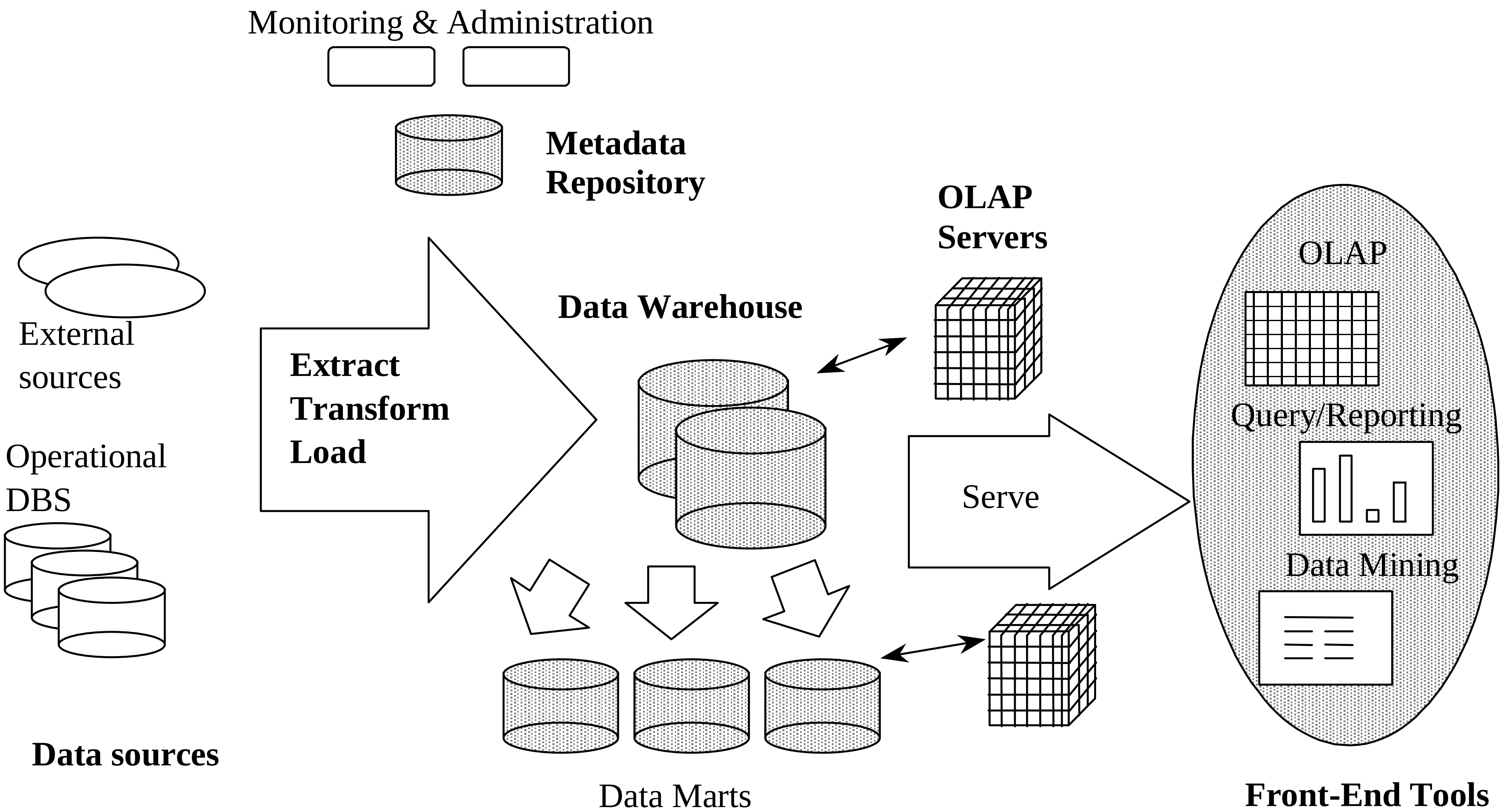
9.2 Association Rules

9.3 Iceberg Queries

9.1 OLAP: Online Analytical Processing

Mining business data for interesting facts and decision support
(CRM, cross-selling, fraud, trading/usage patterns and exceptions, etc.)

- with data from different production sources integrated into ***data warehouse***,
- often with data subsets extracted and transformed into ***data cubes***

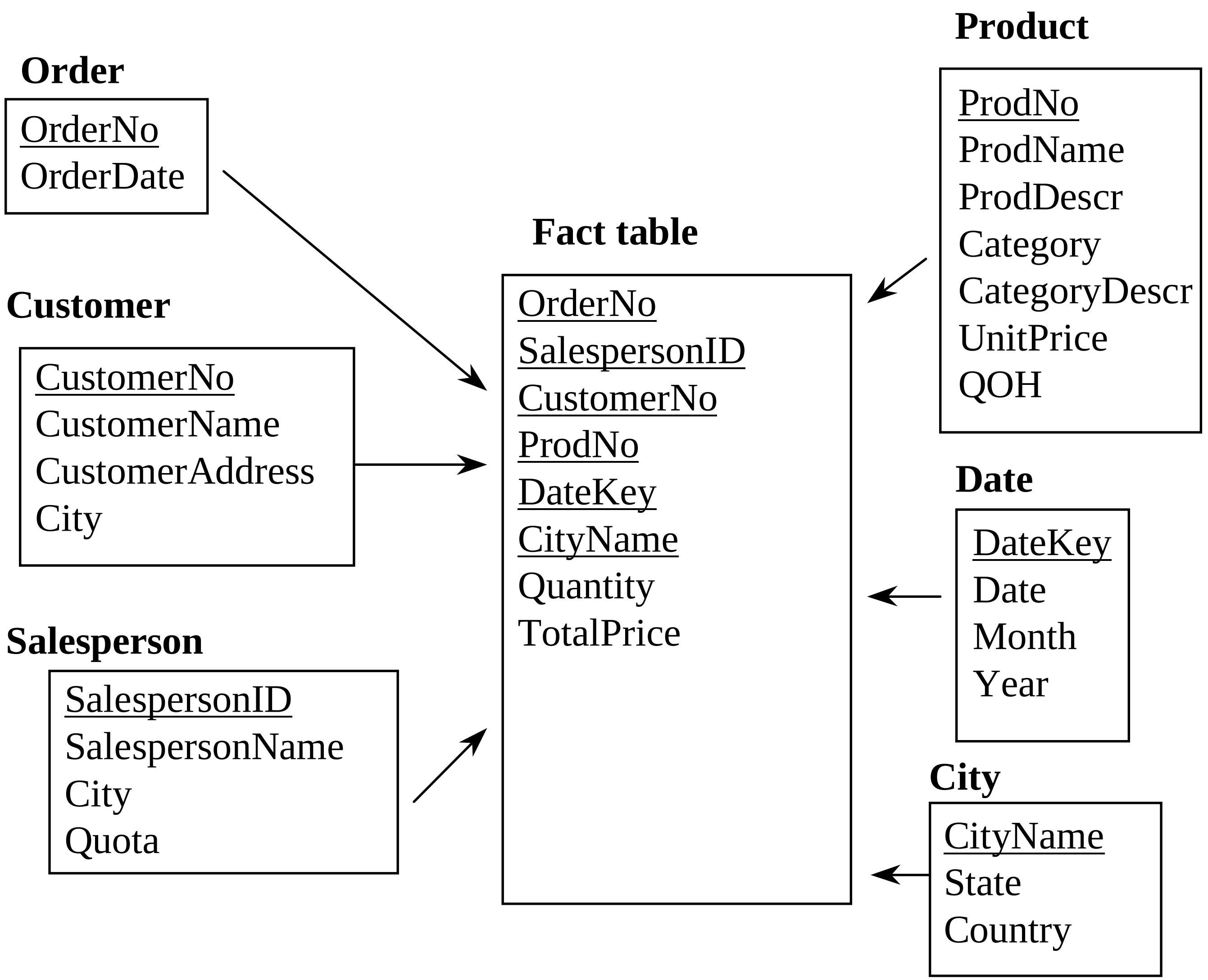


Typical OLAP (Decision Support) Queries

- *What were the sales volumes by region and product category for the last year?*
- *How did the share price of computer manufacturers correlate with quarterly profits over the past 10 years?*
- *Which orders should we fill to maximize revenues?*
- *Will a 10% discount increase sales volume sufficiently?*
- *Which products should we advertise to the various categories of our customers?*
- *Which of two new medications will result in the best outcome: higher recovery rate & shorter hospital stay?*
- *Which ads should be on our Web site to which category of users?*
- *How should we personalize our Web site based on usage logs?*

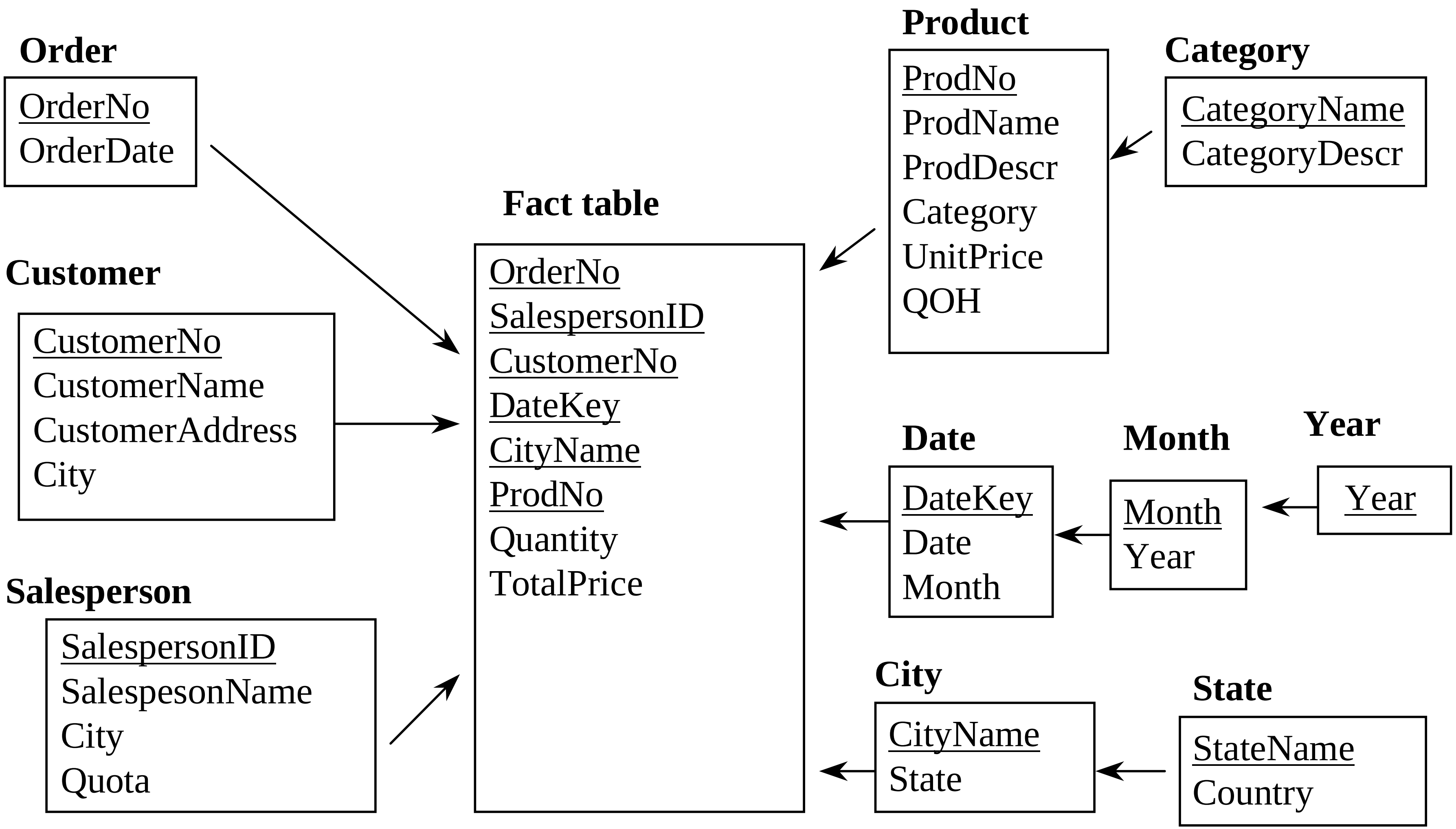
- *Which symptoms indicate which disease?*
- *Which genes indicate high cancer risk?*

Data Warehouse with Star Schema



data often comes from different sources of different organizational units
→ ***data cleaning*** is a major problem

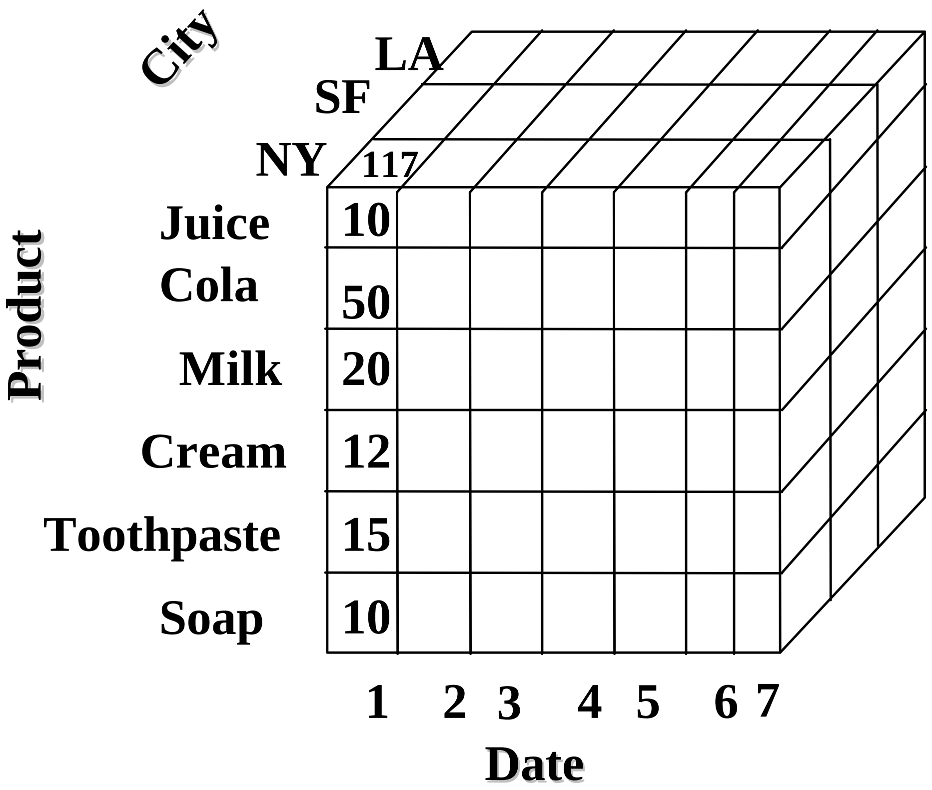
Data Warehouse with Snowflake Schema



Data Cube

- organize data (conceptually) into a multidimensional array
- analysis operations (OLAP algebra, integrated into SQL):
  roll-up/drill-down, slice&dice (sub-cubes), pivot (rotate), etc.

Example: sales volume as a function of product, time, geography



Fact **data: sales volume in $100**

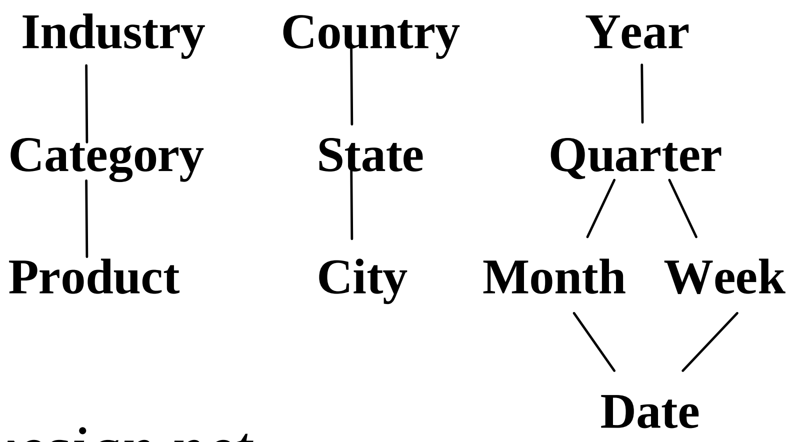
Dimensions:

Product, City, Date

Attributes:

Product (prodno, price, ...)

Attribute Hierarchies and Lattices:



for high dimensionality:
cube could be approximated by Bayesian net

9.2 Association Rules

given:

a set of **items** $I = \{x1, ..., xm\}$

a set (bag) $D=\{t1, ..., tn\}$ of **item sets (transactions)** $ti = \{xi_1, ..., xi_k\} \subseteq I$

wanted:

rules of the form $\mathbf{X \Rightarrow Y}$ with $X \subseteq I$ and $Y \in I$ such that

- X is sufficiently often a subset of the item sets ti and
- when $X \subseteq ti$ then most frequently $Y \in ti$ holds, too.

***support (X ⇒ Y)* = P[XY]** = relative frequency of item sets that contain X and Y

***confidence (X ⇒ Y)* = P[Y|X]** = relative frequency of item sets that contain Y provided they contain X

support is usually chosen in the range of 0.1 to 1 percent,
confidence (aka. strength) in the range of 90 percent or higher

Association Rules: Example

Market basket data („sales transactions“):

t1 = {Bread, Coffee, Wine}
t2 = {Coffee, Milk}
t3 = {Coffee, Jelly}
t4 = {Bread, Coffee, Milk}
t5 = {Bread, Jelly}
t6 = {Coffee, Jelly}
t7 = {Bread, Jelly}
t8 = {Bread, Coffee, Jelly, Wine}
t9 = {Bread, Coffee, Jelly}

support (Bread $\Rightarrow$ Jelly) = 4/9
support (Coffee $\Rightarrow$ Milk) = 2/9
support (Bread, Coffee $\Rightarrow$ Jelly) = 2/9

confidence (Bread $\Rightarrow$ Jelly) = 4/6
confidence (Coffee $\Rightarrow$ Milk) = 2/7
confidence (Bread, Coffee $\Rightarrow$ Jelly) = 2/4

Apriori Algorithm: Idea and Outline

Idea and outline:

- proceed in phases i=1, 2, ..., each making a single pass over D, and generate rules $X \Rightarrow Y$ with frequent item set X (sufficient support) and $|X|=i$ in phase i;
- use phase i-1 results to limit work in phase i:
  antimonotonicity property (downward closedness):
  for i-item-set X to be frequent,
  each subset $X' \subseteq X$ with $|X'|=i-1$ must be frequent, too
- generate rules from frequent item sets;
- test confidence of rules in final pass over D

Worst-case time complexity is exponential in I and linear in D*I,
but usual behavior is linear in D
(detailed average-case analysis is very difficult)

Apriori Algorithm: Pseudocode

procedure apriori (D, min-support):
  L_1 = frequent 1-itemsets(D);
  for (k=2; $L_{k-1} \neq \emptyset$; k++) {
    C_k = apriori-gen (L_{k-1}, min-support);
    for each $t \in D$ { // linear scan of D
      C_t = subsets of t that are in C_k;
      for each candidate $c \in C_t$ {c.count++}; };
    $L_k = \{c \in C_k \mid c.count \geq \text{min-support}\}$; };
  return $L = \cup_k L_k$; // returns all frequent item sets

procedure apriori-gen (L_{k-1}, min-support):
  $C_k = \emptyset$:
  for each itemset $x_1 \in L_{k-1}$ {
    for each itemset $x_2 \in L_{k-1}$ {
      if x_1 and x_2 have k-2 items in common and differ in 1 item // join {
        $x = x_1 \cup x_2$;
        if there is a subset $s \subseteq x$ with $s \notin L_{k-1}$ {disregard x;} // infreq. subset
        else add x to C_k; }; }; };
  return C_k

Algorithmic Extensions and Improvements

- **hash-based counting** (computed during very first pass):
  map k-itemset candidates (e.g. for k=2) into hash table and
  maintain one count per cell; drop candidates with low count early
- **remove transactions** that don‘t contain frequent k-itemset
  for phases k+1, ...
- **partition transactions** D:
  an itemset is frequent only if it is frequent in at least one partition
- **exploit parallelism** for scanning D
- **randomized (approximative) algorithms**:
  find all frequent itemsets with high probability (using hashing etc.)
- **sampling** on a randomly chosen subset of D

...

mostly concerned about reducing disk I/O cost
(for TByte databases of large wholesalers or phone companies)

Extensions and Generalizations of Assocation Rules

- **quantified rules**: consider quantitative attributes of item in transactions (e.g. wine between \$20 and \$50 $\Rightarrow$ cigars, or age between 30 and 50 $\Rightarrow$ married, etc.)
- **constrained rules**: consider constraints other than count thresholds, e.g. count itemsets only if average or variance of price exceeds ...
- **generalized aggregation rules**: rules referring to aggr. functions other than count, e.g., sum(X.price) $\Rightarrow$ avg(Y.age)
- **multilevel association rules**: considering item classes (e.g. chips, peanuts, bretzels, etc. belonging to class snacks)
- **sequential patterns** (e.g. an itemset is a customer who purchases books in some order, or a tourist visiting cities and places)
- from strong rules to **interesting rules**: consider also lift (aka. interest) of rule $X \Rightarrow Y$: $P[XY] / P[X]P[Y]$
- **correlation rules**
- **causal rules**

Correlation Rules

example for strong, but misleading association rule:

tea $\Rightarrow$ coffee with confidence 80% and support 20%

but support of coffee alone is 90%, and of tea alone it is 25%

$\rightarrow$ tea and coffee have negative correlation !

consider contingency table (assume n=100 transactions):

	T	¬T	
C	20	70	90
¬C	5	5	10
	25	75	

$\rightarrow$ {T, C} is a frequent and correlated item set

$$\chi^2(C,T) = \sum_{X \in \{C,\overline{C}\}} \sum_{Y \in \{T,\overline{T}\}} (\frac{(freq(X \wedge Y) - freq(X)\,freq(Y)/n)^2}{freq(X)\,freq(Y)/n}$$

correlation rules are **monotone (upward closed):**

if the set X is correlated then every superset X‘ $\supseteq$ X is correlated, too.

Correlation Rules

example for strong, but misleading association rule:

tea $\Rightarrow$ coffee with confidence 80% and support 20%

but support of coffee alone is 90%, and of tea alone it is 25%

$\rightarrow$ tea and coffee have negative correlation !

consider contingency table (assume 100 transactions):

	T	$\neg$T	
C	20	70	90
$\neg$C	5	5	10
	25	75	

E[C]=0.9

E[T]=0.25

E[(T-E[T])2]=1/4 * 9/16 +3/4 * 1/16= 3/16=Var(T)

E[(C-E[C])2]=9/10 * 1/100 +1/10 * 1/100 = 9/100=Var(C)

E[(T-E[T])(C-E[C])]=

2/10 * 3/4 * 1/10

- 7/10 * 1/4 * 1/10

- 5/100 * 3/4 * 9/10

+ 5/100 * 1/4 * 9/10 =

60/4000 – 70/4000 – 135/4000 + 45/4000 = - 1/40 = Cov(C,T)

ρ(C,T) = - 1/40 * 4/sqrt(3) * 10/3 ≈ -1/(3*sqrt(3)) ≈ - 0.2

Correlated Item Set Algorithm

procedure corrset (D, min-support, support-fraction, significance-level):

```
for each x ∈ I compute count O(x);
initialize candidates := ∅; significant := ∅;
for each item pair x, y ∈ I with O(x) > min-support and O(y) > min-support {
    add (x,y) to candidates; };
while (candidates ≠ ∅) {
    notsignificant := ∅;
    for each itemset X ∈ candidates {
        construct contingency table T;
        if (percentage of cells in T with count > min-support
        is at least support-fraction) { // otherwise too few data for chi-square
            if (chi-square value for T ≥ significance-level)
            {add X to significant} else {add X to notsignificant};
        }; //if
    }; //for
    candidates := itemsets with cardinality k such that
                  every subset of cardinality k-1 is in notsignificant;
                  // only interested in correlated itemsets of min. cardinality
}; //while
return significant
```

9.3 Iceberg Queries

Queries of the form:

Select A1, ..., Ak, aggr(Arest) From R
Group By A1, ..., Ak Having aggr(Arest) >= threshold

with some aggregation function aggr (often count(*));
A1, ..., Ak are called targets, (A1, ..., Ak) with an aggr value above the threshold is called a frequent target

Baseline algorithms:
1) scan R and maintain aggr field (e.g. counter) for each (A1, ..., Ak) or
2) sort R, then scan R and compute aggr values

but: 1) may not be able to fit all (A1, ..., Ak) aggr fields in memory
2) has to scan huge disk-resident table multiple times

Iceberg queries are very useful as an efficient building block in algorithms for rule generation, interesting-fact or outlier detection (on market baskets, Web logs, time series, sensor streams, etc.)

Examples for Iceberg Queries

Market basket rules:

Select Part1, Part2, Count(*) From All-Coselling-Part-Pairs
Group By Part1, Part2 Having Count(*) >= 1000

Select Part, Region, Sum(Quantity * Price) From OrderLineItems
Group By Part, Region Having Sum(Quantity*Price) >= 100 000

Frequent words (stopwords) or frequent word pairs in docs

Overlap in docs for (mirrored or pirate) copy detection:

Select D1.Doc, D2.Doc, Count(D1.Chunk)
From DocSignatures D1, DocSignatures D2
Where D1.Chunk = D2.Chunk And D1.Doc != D2.Doc
Group By D1.Doc, D2.Doc Having Count(D1.Chunk) >= 30

table R should avoid materialization of all (doc chunk) pairs

Acceleration Techniques

V: set of targets, |V|=n, |R|=N, V[r]: r^{th} most frequent target
H: heavy targets with freq. $\geq$ threshold t, |H|=max{r | V[r] has freq. $\geq$ t}
L = V-H: light targets, F: potentially heavy targets

Determine F by sampling
scan s random tuples of R and compute counts for each $x \in V$;
if freq(x) $\geq$ t * s/N then add x to F
or by „coarse“ (probabilistic) counting
scan R, hash each $x \in V$ into memory-resident table A[1..m], m<n;
scan R, if A[h(x)] $\geq$ t then add x to F

Remove false positives from F (i.e., $x \in F$ with $x \in L$)
by another scan that computes exact counts only for $\in F$

Compensate for false negatives (i.e., $x \notin F$ with $x \in H$)
e.g. by removing all $H' \subset H$ from R and doing an exact count
(assuming that some $H' \subset H$ is known, e.g. „superheavy“ targets)

Defer-Count Algorithm

Key problem to be tackled:
coarse-counting buckets may become heavy
by many light targets or by few heavy targets or combinations

1) Compute small sample of s tuples from R;
Select f potentially heavy targets from sample and add them to F;
2) Perform coarse counting on R, ignoring all targets from F
(thus reducing the probability of false positives);
Scan R, and add targets with high coarse counts to F;
3) Remove false positives by scanning R and doing exact counts

Problems:
difficult to choose values for tuning parameters s and f
phase 2 divides memory between initial F and hash table for counters

Multi-Scan Defer-Count Algorithm

1) Compute small sample of s tuples from R;
   Select f potentially heavy targets from sample and add them to F;
2) for i=1 to k with independent hash functions $h_1, ..., h_k$ do
   perform coarse counting on R using h_i, ignoring targets from F;
   construct bitmap B_i with $B_i[j]=1$ if j-th bucket is heavy
3) scan R and add x to F if $B_i[h_i(x)]=1$ for all i=1, ..., k;
4) remove false positives by scanning R and doing exact counts

+ further optimizations and combinations with other techniques

Multi-Level Algorithm

1) Compute small sample of s tuples from R;
   Select f potentially heavy targets from sample and add them to F;
2) Initialize hash table A:
   mark all h(x) with $x \in F$ as potentially heavy and
   allocate m‘ auxiliary buckets for each such h(x);
   set all entries of A to zero
3) Perform coarse counting on R:
   if h(x) is not marked then increment h(x) counter
   else increment counter of h‘(x) auxiliary bucket
   using a second hash function h‘;
   scan R, and add targets with high coarse counts to F;
4) Remove false positives by scanning R and doing exact counts

Problem:
how to divide memory between A and the auxiliary buckets

Iceberg Query Algorithms: Example

R = {1, 2, 3, 4, 1, 1, 2, 4, 1, 1, 2, 4, 1, 1, 2, 4, 1, 1, 2, 4}, N=20
threshold T=8 → H={1}
hash function h_1: dom(R) → {0,1}, $h_1(1)=h_1(3)=0$, $h_1(2)= h_1(4)=1$,
hash function h_2: dom(R) → {0,1}, $h_2(1)=h_2(4)=0$, $h_2(2)=h_2(3)=1$,

Defer-Count:
s=5 → F={1}
using h1: cnt(0)=1, cnt(1)=10
bitmap 01, re-scan → F={1, 2, 4}
final scan with exact counting
→ H={1}

Multi-scan Defer-Count:
s=5 → F={1}
using h1: cnt(0)=1, cnt(1)=10
using h2: cnt(0)=5, cnt(1)=6
re-scan → F={1}
final scan with exact counting
→ H={1}

Additional Literature for Chapter 9

- J. Han, M. Kamber, Chapter 6: Mining Association Rules
- D. Hand, H. Mannila, P. Smyth: Principles of Data Mining, MIT Press, 2001, Chapter 13: Finding Patterns and Rules
- M.H. Dunham, Data Mining, Prentice Hall, 2003, Ch. 6: Association Rules
- M. Ester, J. Sander, Knowledge Discovery in Databases, Springer, 2000, Kapitel 5: Assoziationsregeln, Kapitel 6: Generalisierung
- M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman: Computing Iceberg Queries Efficiently, VLDB 1998
- S. Brin, R. Motwani, C. Silverstein: Beyond Market Baskets: Generalizing Association Rules to Correlations, SIGMOD 1997
- C. Silverstein, S. Brin, R. Motwani, J.D. Ullman: Scalable Techniques for Mining Causal Structures, Data Mining and Knowledge Discovery 4(2), 2000
- R.J. Bayardo: Efficiently Mining Long Patterns from Databases, SIGMOD 1998
- D. Margaritis, C. Faloutsos, S. Thrun: NetCube: A Scalable Tool for Fast Data Mining and Compression, VLDB 2001
- R. Agrawal, T. Imielinski, A. Swami: Mining Association Rules Between Sets of Items in Large Databases, SIGMOD 1993
- T. Imielinski, Data Mining, Tutorial, EDBT Summer School, 2002, http://www-lsr.imag.fr/EDBT2002/Other/edbt2002PDF/EDBT2002School-Imielinski.pdf
- R. Agrawal, R. Srikant, Whither Data Mining?, http://www.cs.toronto.edu/vldb04/Agrawal-Srikant.pdf