# Association
# (Frequent Itemset Mining)

# Association

- Association mining : finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
  - Market basket analysis(MBA)

- Motivation: finding regularities in data
  - what products were often purchased together?
    * Beer and diapers on Thursday night?!
  - what are the subsequent purchases after buying a PC?
  - what kinds of DNA are sensitive to this new drug?

- Although association rules were originally derived from point-of-sale data that describes what products are purchased together, they can be applied to find relationships among other types of databases.

# Association Rules

- Association rules are easy to understand, but not always useful
  - Wall-Mart customers who purchase Barbie dolls have a 60% likelihood of also purchasing candy bars (from Forbes. actionable rule)
  - customers who purchase maintenance agreements are very likely to purchase large appliances (trivial)
  - when a new hardware store opens, one of the most commonly sold items is toilet bowl cleaners (inexplicable)

- When applying association, many of the results are often either trivial or inexplicable.
- Famous rules: beer and diaper
  - on Thursdays, grocery store consumers often purchase diapers and beer together
  - young couples prepare for the weekend by stocking up on diapers for the infants and beer for dad
    * managers can now take action

# Association

- Potential application
  - items purchased on a credit card give insight into the next product that customers are likely to purchase
  - optional services(call waiting, fall forwarding, ISDN, etc.) purchased by telecommunications customers help determine how to bundle these services together to maximize revenue
  - banking services(CDs, investment services, car loans, etc.) used by retail customers identify customers likely to want other services
  - unusual combinations of insurance claims can be a sign of fraud and can spark further investigation
  - medical patient histories can give indications of complications based on certain combination of treatments

# Item Sets: A New Type of Data

- Set of items: $I=\{I_1,I_2,\ldots,I_m\}$
- Database: $T = \{t1, t2,.., tn\}$ be a set of transactions, where a transaction *ti* is a set of items
- Transaction: $ti=(tid,X)$
  - tid: transaction identifier
  - Itemset: $X \subseteq I$

| *TID* | *Items* |
|---|---|
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer, Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

# Definitions

- **Itemset**
  - A collection of one or more items
  - Example: {Milk, Bread, Diaper}
- **k-itemset**
  - An itemset that contains k items
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* threshold
  - For example, if threshold is 3, then {bread, milk} is a frequent itemset as it occurs in 3 or more transactions in the dataset.

| *TID* | *Items* |
|---|---|
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer, Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

# Definitions

- Association Rule:
  - Expression of the form X -> Y (X and Y are itemsets)
  - Example: {Milk, Diaper} -> Beer

- Rule Evaluation measures
  - **Support**(s): Fraction of transactions that contain both X and Y
  - **Confidence**(c): measures how often items in Y appear in transactions that contain X

| *TID* | *Items* |
|---|---|
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer, Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

- Example:
  - {Milk, Diaper} -> Beer
  - s=support(Milk, Diaper, Beer)/|T| =2/5 = 0.4
  - c=support(Milk, Diaper, Beer)/support(Milk, Diaper) =2/3 =0.67

# Support and Confidence

- If a customer purchases soda, then the customer also purchases milk: {soda ->milk}
  - how good is this rule ?

- Support
  - fraction of transactions that contain both left-hand and right-hand
  - two of the five transaction include both soda and OJ
  - support = 2/5

$$Support(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

- Confidence
  - the ratio of the number of the transactions with all the items to the number of transactions with just the "if" items.
  - in case "if soda, then OJ" confidence = 2/3
  - in case "if OJ, then soda" confidence = 2/4

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

**Table 9.1** Grocery Point-of-Sale Transactions

| CUSTOMER | ITEMS |
|---|---|
| 1 | Orange juice, soda |
| 2 | Milk, orange juice, window cleaner |
| 3 | Orange juice, detergent |
| 4 | Orange juice, detergent, soda |
| 5 | Window cleaner, soda |

# Support

- Suppose the most common combination has three items. The combinations of three items, A, B, and C are
  - If A and B, then C:  AB->C
  - If A and C, then B: AC->B
  - If B and C, then A: BC->A
  - these three rules have the same support(5%) (in Tab. 9.6)
    - p(condition and result) means support

**Table 9.1** Grocery Point-of-Sale Transactions

| CUSTOMER | ITEMS |
|---|---|
| 1 | Orange juice, soda |
| 2 | Milk, orange juice, window cleaner |
| 3 | Orange juice, detergent |
| 4 | Orange juice, detergent, soda |
| 5 | Window cleaner, soda |

**Table 9.6** Confidence in Rules

| RULE | P(CONDITION) | P(CONDITION AND RESULT) | CONFIDENCE |
|---|---|---|---|
| If A and B then C | 25% | 5% | 0.20 |
| If A and C then B | 20% | 5% | 0.25 |
| If B and C then A | 15% | 5% | 0.33 |

# Association Rule Mining

- **Association Rule Mining**: Given set of transactions T, find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

- Example (s: support, c: confidence)

| *TID* | *Items* |
|---|---|
| 1 | Bread, Milk |
| 2 | Bread, Diaper, Beer, Eggs |
| 3 | Milk, Diaper, Beer, Coke |
| 4 | Bread, Milk, Diaper, Beer |
| 5 | Bread, Milk, Diaper, Coke |

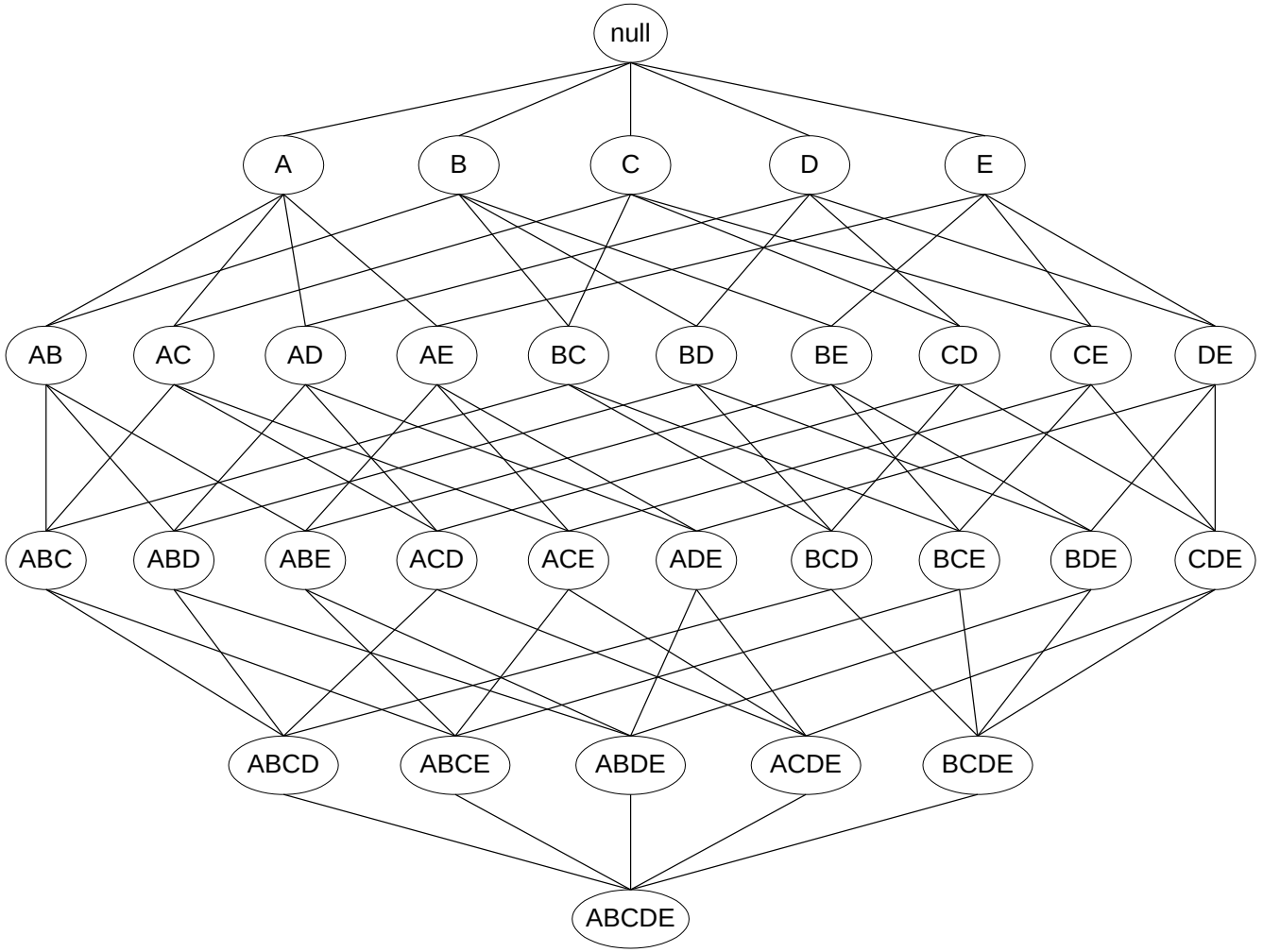
{Milk,Diaper} -> {Beer } (s=0.4, c=0.67)
{Milk,Beer} -> {Diaper } (s=0.4, c=1.0)
{Diaper,Beer} -> {Milk } (s=0.4, c=0.67)
{Beer} -> {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} -> {Milk,Beer} (s=0.4, c=0.5)
{Milk} -> {Diaper,Beer} (s=0.4, c=0.5)
...

# Brute Force Method

- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
    ⇒ Computationally prohibitive (combinatorial explosion)

- Brute force:
  Match every itemset against transaction database

- Given i items, there are $2^i - 1$ candidate itemsets



# Brute Force Method

- Can we do better? (reduce complexity)

- Observations:
  - All rules are binary partitions of an itemset: e.g., {Milk, Diaper, Beer}
  - Rules originating from the same itemset have **identical support** but can have **different confidence**
  - Thus, we may **decouple** the support and confidence requirements

- Many techniques haven been proposed to reduce the complexity

# Association Rule Mining

- Two-step approach:
  - Frequent Itemset Generation (count support)
    - Generate *all* frequent itemsets whose support ≥ *minsup*
  - Rule Generation (count confidence)
    - Generate rules whose confidence ≥ *minconf* from each frequent itemset, where each rule is a *binary partitioning* of a frequent itemset

- To reduce the complexity of the method, we need to:
  1) Reduce the number of frequent itemsets(candidates) (M)
     - *Pruning*
  2) Reduce the number of comparisons (to compute support) in transactions (NM)
     - *Efficient support counting*

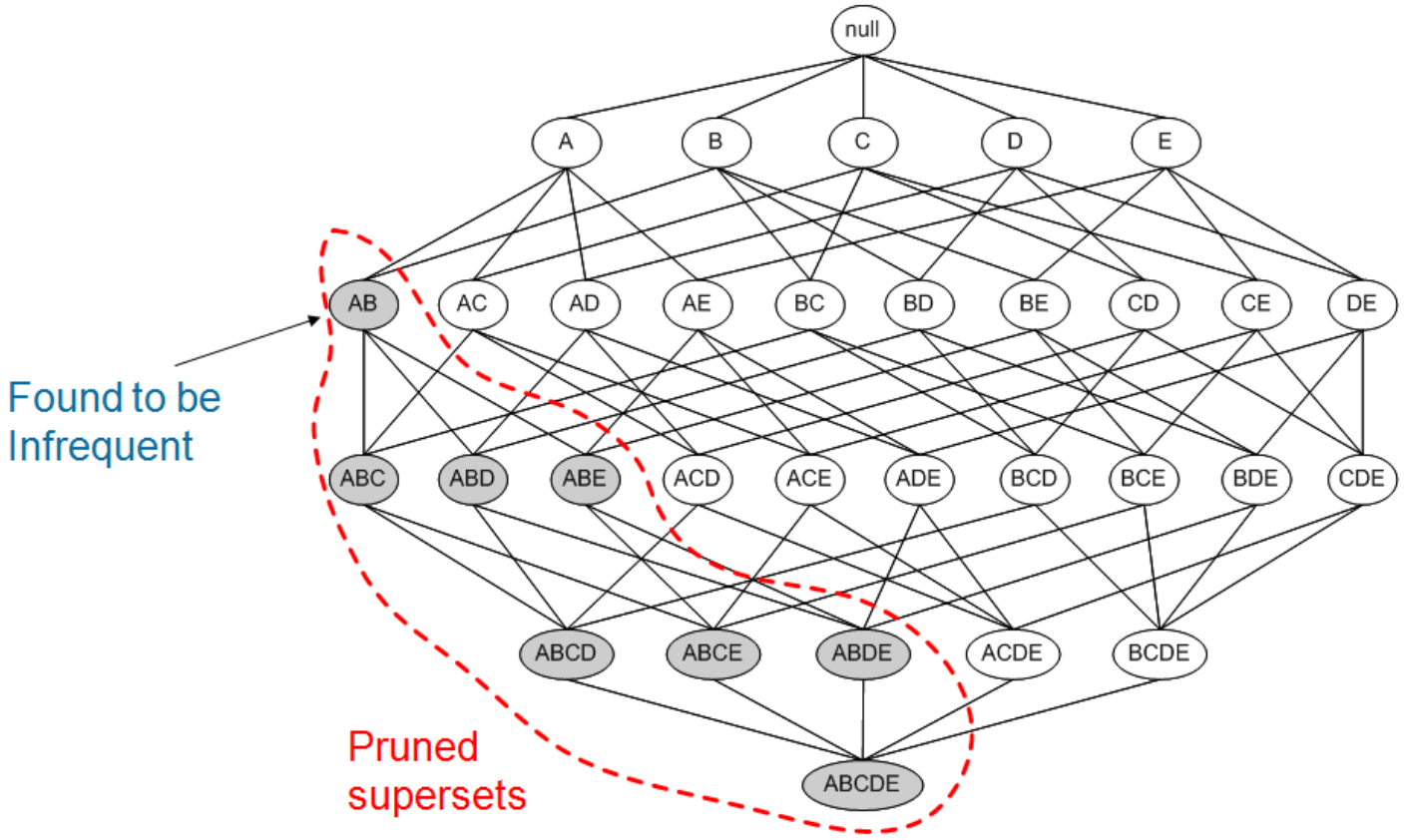
# Reduce Number of Frequent Itemsets

- Apriori principle
  - if an itemset is frequent, then all of its subsets must also be frequent

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

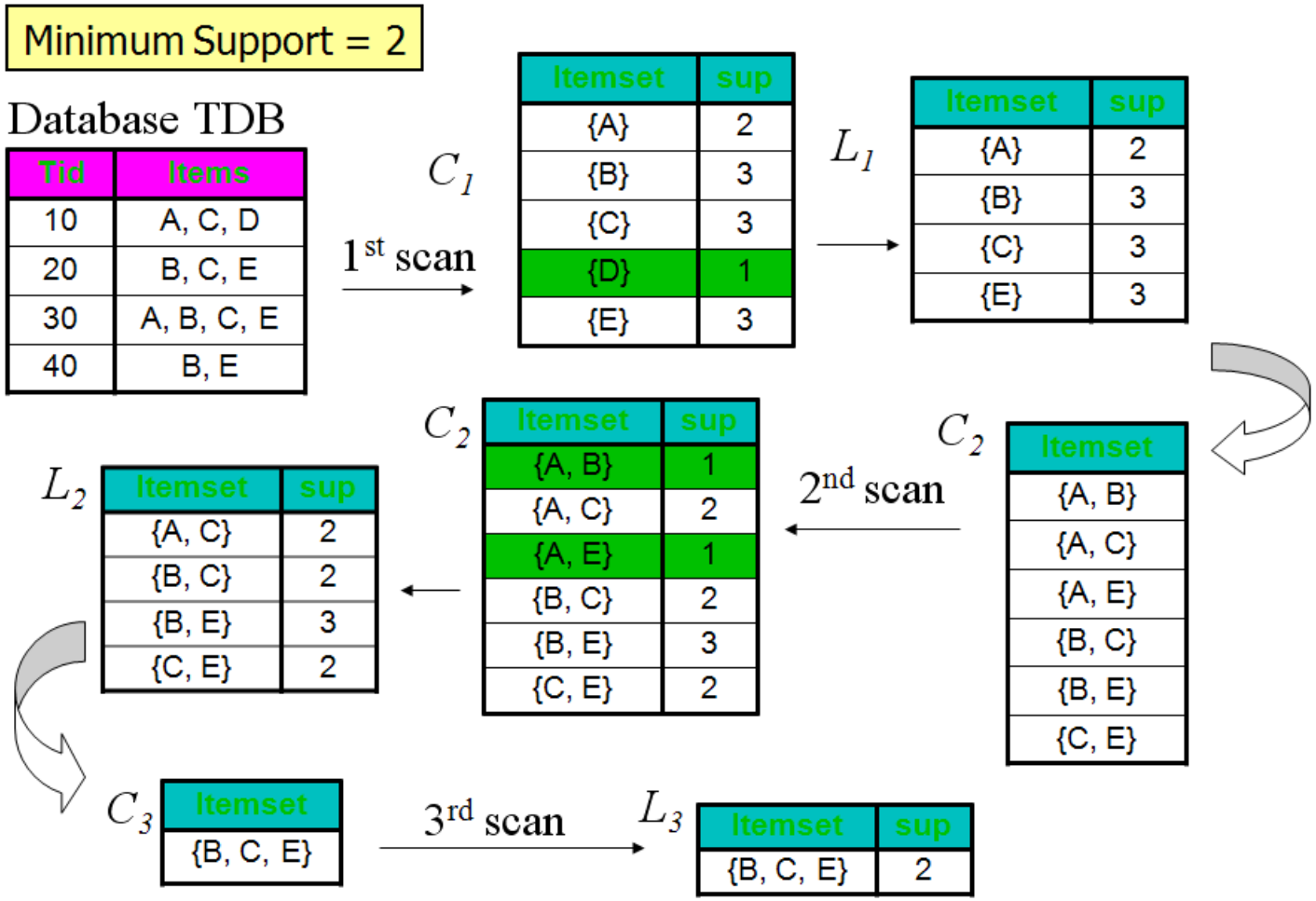
where X, Y represent itemsets.

  - **support of an itemset never exceeds the support of its subsets**
  - Anti-monotonicity



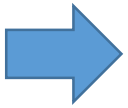
# Apriori Algorithm

- Apriori algorithm: generate frequent itemsets using Apriori principle

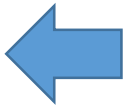


# Apriori Algorithm

| candidate | support | Frequent? |
|---|---|---|
| A | 6 | Y |
| B | 7 | Y |
| C | 6 | Y |
| D | 2 | Y |
| E | 2 | Y |



| candidate | support | Frequent? |
|---|---|---|
| AB | 4 | Y |
| AC | 4 | Y |
| ~~AD~~ | ~~1~~ | ~~N~~ |
| AE | 2 | Y |
| BC | 4 | Y |
| BD | 2 | Y |
| BE | 2 | Y |
| ~~CD~~ | ~~0~~ | ~~N~~ |
| ~~CE~~ | ~~1~~ | ~~N~~ |
| ~~DE~~ | ~~0~~ | ~~N~~ |



| candidate | support | Frequent? |
|---|---|---|
| ABC | 2 | Y |
| ABE | 2 | Y |

| TID | Items |
|---|---|
| 1 | ABE |
| 2 | BD |
| 3 | BC |
| 4 | ABD |
| 5 | AC |
| 6 | BC |
| 7 | AC |
| 8 | ABCE |
| 9 | ABC |

min support = 2

# Apriori Algorithm

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the DB
  - Eliminate infrequent candidates, leaving only those that are frequent

# Reduce Number of Comparisons

| candidate | support | Frequent? |
|---|---|---|
| AB | 4 | Y |
| AC | 4 | Y |
| ~~AD~~ | ~~1~~ | ~~N~~ |
| AE | 2 | Y |
| BC | 4 | Y |
| BD | 2 | Y |
| BE | 2 | Y |
| ~~CD~~ | ~~0~~ | ~~N~~ |
| ~~CE~~ | ~~1~~ | ~~N~~ |
| ~~DE~~ | ~~0~~ | ~~N~~ |

| TID | Items |
|---|---|
| 1 | ABE |
| 2 | BD |
| 3 | BC |
| 4 | ABD |
| 5 | AC |
| 6 | BC |
| 7 | AC |
| 8 | ABCE |
| 9 | ABC |

| Key | Value |
|---|---|
| A,C,E | AB, AC, AD, AE, CD, CE |
| B,D | BC, BD, BE, DE |

- Another method to reduce complexity is to use a hash function to store transaction databases
- Efficient data structure to look up items in a list

# Back to Association Rules

- Given a frequent itemset L, find all non-empty subsets f ⊂ L such that f → L – f satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules
  - ABC →D, ABD →C, ACD →B, BCD →A, A →BCD, B →ACD, C →ABD, D →ABC, AB →CD, AC → BD, AD → BC, BC →AD, BD →AC, CD →AB
  - If |L| = k, then there are $2^k - 2$ candidate association rules (ignoring L → ∅ and ∅ → L)

- Example: association rules from frequent itemset { Milk, Diaper, Beer }

{ Milk, Diaper, Beer } →

{Milk,Diaper} -> {Beer } (s=0.4, c=0.67)
{Milk,Beer} -> {Diaper } (s=0.4, c=1.0)
{Diaper,Beer} -> {Milk } (s=0.4, c=0.67)
{Beer} -> {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} -> {Milk,Beer} (s=0.4, c=0.5)
{Milk} -> {Diaper,Beer} (s=0.4, c=0.5)

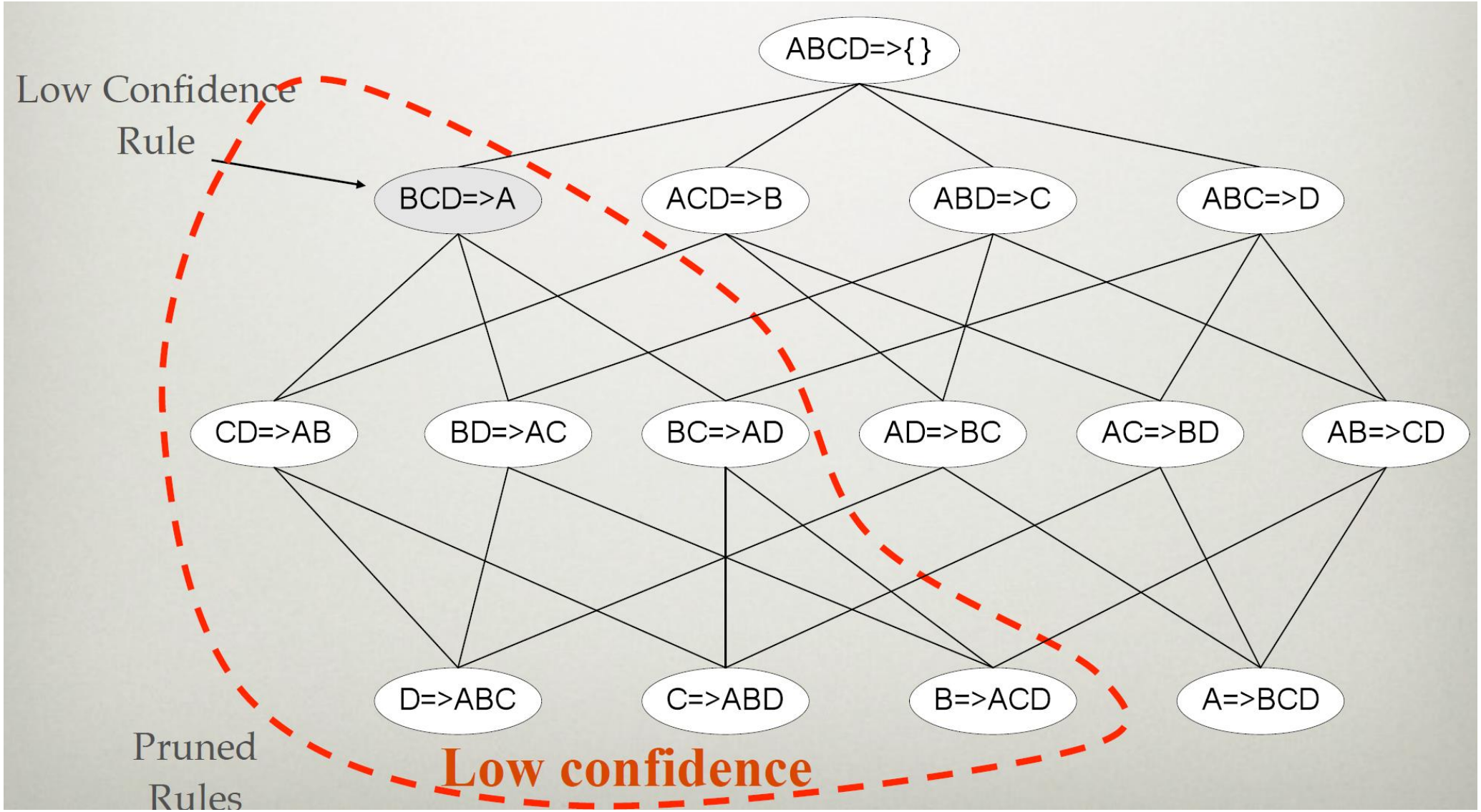
# Rule Generation from a Frequent Itemset

- How to efficiently generate rules from a frequent itemset?

- Unlike Apriori principle in support, **confidence** is *not* anti-monotone
  - c(ABC →D) can be larger or smaller than c(AB →D)

- **Confidence of rules** generated from the **same** itemset *is*!
  - e.g., L = {A,B,C,D}:
    - c(ABC → D) ≥ c(AB → CD) ≥ c(A → BCD)
    - Anti-monotone w.r.t. number of items on the RHS of the rule in the same itemset.

# Rule Generation from a Frequent Itemset



# Lift

- In Tab. 9.6, the most confident rule is the best rule
  - so "If B and C, then A" is the best ? (s=5%, c=33%)

- Consider the rule "If null, then A" (randomly saying A appears in the transaction)
  - support = 45%
  - confidence = 45%
  - *better than above rule*

**Table 9.6** Confidence in Rules

| RULE | P(CONDITION) | P(CONDITION AND RESULT) | CONFIDENCE |
|---|---|---|---|
| If A and B then C | 25% | 5% | 0.20 |
| If A and C then B | 20% | 5% | 0.25 |
| If B and C then A | 15% | 5% | 0.33 |

# Lift

- Lift(improvement) : For a rule X->Y, how much better a rule is at predicting the result than just assuming the right hand side.
  - lift = $p(Y|X) / p(Y) = support(X \rightarrow Y)/support(\emptyset \rightarrow Y)$
    $= support(X \cup Y)/(support(X)*support(Y))$

$$= \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

  - greater than 1 : the resulting rule is better at predicting the result than random chance
  - less than 1 : it is worse than random chance

- None of the rules with three items shows improved lift
  - the best rule has two items

**Table 9.7** Lift Measurements for Four Rules

| RULE | SUPPORT | CONFIDENCE | P(RESULT) | LIFT |
|---|---|---|---|---|
| If A and B then C | 5% | 0.20 | 40% | 0.50 |
| If A and C then B | 5% | 0.25 | 42.5% | 0.59 |
| If B and C then A | 5% | 0.33 | 45% | 0.74 |
| If A then B | 25% | 0.59 | 42.5% | 1.31 |

# Dissociation (Negative) Rules

- When lift is less than 1, negating the result produces a better rule
  - If B and C, then A (confidence=0.33)
  - If B and C, then NOT A (confidence=0.67)
- Dissociation rule : similar to association rule except that it can have the connector "and not" in the condition in addition to "and."
  - If A and not B then C
- Modification of transaction database
  - introduce a new set of items that are the inverses of each of the original items
  - modify each transaction so it includes an inverse item iff it does not contain the original item

**Table 9.8** Transformation of Transactions to Generate Dissociation Rules

| CUSTOMER | ITEMS | CUSTOMER | WITH INVERSE ITEMS |
|---|---|---|---|
| 1 | {A, B, C} | 1 | {A, B, C} |
| 2 | {A} | 2 | {A, ¬B, ¬C} |
| 3 | {A, C} | 3 | {A, ¬B, C} |
| 4 | {A} | 4 | {A, ¬B, ¬C} |
| 5 | {} | 5 | {¬A, ¬B, ¬C} |

# Dissociation Rules

- Three downsides
  1) the total number of items doubles and computation grows exponentially
  2) the size of transaction typical transaction grows
  3) the frequency of the inverse items tends to be much larger than the frequency of the original items.

- Minimum support constraints tend to produce rules in which all items are inverted
  - If not A and not B then not C
  - less likely to be actionable

# Multi-dimensional Association

- So far we have applied association rule mining in transaction databases.

- How to apply association rule mining to general multi-dimensional relational tables ?

- Change relational tables to transaction database
- Introduce a new "item" for each distinct attribute-value pair (One-hot-encoding)

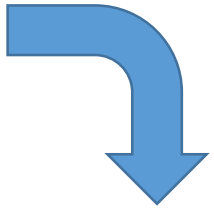
  - Example: replace ‘Temperature’ with 3 binary attributes
    {Temperature = ‘hot’, Temperature = ‘mild’, Temperature = ‘cool’ }
  - Example of multi-dimensional Association Rule:
    {Outlook=‘sunny’ & Humidity=‘normal’} -> {Play=‘Yes’}

- We can apply *Association* method to relational databases

- Need techniques for treating *continuous* and *categorical* attributes.
  - Transform categorical attribute into one-hot-encoding
  - Discretize continuous attributes

# Convert Relational Table to Transaction Data

| Outlook | Temperature | Humidity | Play? |
|---|---|---|---|
| sunny | hot | high | No |
| sunny | hot | high | No |
| cloudy | hot | high | Yes |
| rain | mild | high | Yes |
| rain | cool | normal | Yes |
| rain | cool | normal | No |



**One-hot-vector encoding**

| Outlook | | | Temperature | | | Humidity | | Play? | |
|---|---|---|---|---|---|---|---|---|---|
| sunny | cloudy | rain | hot | mild | cool | high | normal | Yes | No |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

# Multi-dimensional Association

- Example: Congressional Voting Records
- The data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes in 1984
- Attribute information
  1. Class Name: 2 (democrat, republican)
  2. handicapped-infants: 2 (y,n)
  3. water-project-cost-sharing: 2 (y,n)
  4. budget-resolution: 2 (y,n)
  5. physician-fee-freeze: 2 (y,n)
  6. el-salvador-aid: 2 (y,n)
  7. religious-groups-in-schools: 2 (y,n)
  8. anti-satellite-test-ban: 2 (y,n)
  9. aid-to-nicaraguan-contras: 2 (y,n)
  10. mx-missile: 2 (y,n)
  11. immigration: 2 (y,n)
  12. synfuels-corporation-cutback: 2 (y,n)
  13. education-spending: 2 (y,n)
  14. right-to-sue: 2 (y,n)
  15. crime: 2 (y,n)
  16. duty-free-exports: 2 (y,n)
  17. export-administration-act-south-africa: 2 (y,n)

# Multi-dimensional Association

- Multi-dimensional Association
  - needs massaging for association
  - 17-feature set
  - 435 transactions
  - Applied Apriori algorithm
  - Minsup=30%, minconf=90%

- Resulting Association Rules
  - (budget-resolution =n, MX-missile=n, el-salvador-aid=y) -> republican
  - (budget-resolution =y, MX-missile=y, el-salvador-aid=n) -> democrat
  - (crime=y, right-to-sue=y, physician-fee-freeze=y) -> republican
  - (crime=n, right-to-sue=n, physician-fee-freeze=n) -> democrat

# Handling Continuous Attributes

- Different kinds of rules:
  - Age ∈ [21,35) & Salary∈ (70k,120k) -> Buy

- Unsupervised discretization method (refer to data preprocessing chapter)
  - Equal-width binning, Equal-depth binning, Clustering (k means)

- Supervised discretization method (refer to data preprocessing chapter)
  - Entropy-based, Chi-square

- Size of the discretized intervals affect support & confidence

$$\{\text{Refund} = \text{No}, (\text{Income} = \$51{,}250)\} \rightarrow \{\text{Cheat} = \text{No}\}$$

$$\{\text{Refund} = \text{No}, (60\text{K} \leq \text{Income} \leq 80\text{K})\} \rightarrow \{\text{Cheat} = \text{No}\}$$

$$\{\text{Refund} = \text{No}, (0\text{K} \leq \text{Income} \leq 1\text{B})\} \rightarrow \{\text{Cheat} = \text{No}\}$$

  - If intervals too small:
    * may not have enough support
  - If intervals too large
    * may not have enough confidence

# Apriori in Python

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth, association_rules

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
     ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],   ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
     ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],   ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)

# frequent itemsets
print(frequent_itemsets)
# generate association rules
print(association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7))
```

# Apriori in Python

frequent itemsets

| | | |
|---|---|---|
| 0 | 0.8 | (Eggs) |
| 1 | 1.0 | (Kidney Beans) |
| 2 | 0.6 | (Milk) |
| 3 | 0.6 | (Onion) |
| 4 | 0.6 | (Yogurt) |
| 5 | 0.8 | (Kidney Beans, Eggs) |
| 6 | 0.6 | (Onion, Eggs) |
| 7 | 0.6 | (Kidney Beans, Milk) |
| 8 | 0.6 | (Kidney Beans, Onion) |
| 9 | 0.6 | (Kidney Beans, Yogurt) |
| 10 | 0.6 | (Kidney Beans, Onion, Eggs) |

association rules

| | antecedents | consequents | support | ... |
|---|---|---|---|---|
| 0 | (Kidney Beans) | (Eggs) | ... | |
| 1 | (Eggs) | (Kidney Beans) | ... | |
| 2 | (Onion) | (Eggs) | ... | |
| 3 | (Eggs) | (Onion) | ... | |
| 4 | (Milk) | (Kidney Beans) | ... | |
| 5 | (Onion) | (Kidney Beans) | ... | |
| 6 | (Yogurt) | (Kidney Beans) | ... | |
| 7 | (Kidney Beans, Onion) | (Eggs) | ... | |
| 8 | (Kidney Beans, Eggs) | (Onion) | ... | |
| 9 | (Onion, Eggs) | (Kidney Beans) | ... | |
| 10 | (Onion) | (Kidney Beans, Eggs) | ... | |
| 11 | (Eggs) | (Kidney Beans, Onion) | ... | |

Demo: https://www.philippe-fournier-viger.com/tools/Apriori_algorithm_demo.php

# Bottleneck of Frequent-Pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of Candidates: = $2^{100}-1 = 1.27*10^{30}$
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

# Mining Freq Patterns w/o Candidate Generation

- Compress a large database into a compact Frequent-Pattern tree (FP-tree) structure
  - Highly condensed, but complete for frequent pattern mining
  - Avoid costly database scans

- Develop an efficient, FP-tree-based frequent pattern mining method
  - Avoid candidate generation: examine sub-database (conditional pattern base) only!

# FP-Tree and FP-Growth Algorithm

- FP-Tree: Frequent Pattern Tree
  - Compact presentation of the DB without information loss.
  - Easy to traverse, can quickly find out patterns associated with a certain item.
  - Well-ordered by item frequency.

- FP-Growth Algorithm
  - Start mining from length-1 patterns
  - Recursively do the following
    - Constructs its conditional FP-tree
    - Concatenate patterns from conditional FP-tree with suffix
  - Divide-and-Conquer mining technique

# FP-Tree Construction

| Trans ID | Items |
|---|---|
| T1 | {E,K,M,N,O,Y} |
| T2 | {D,E,K,N,O,Y} |
| T3 | {A,E,K,M} |
| T4 | {C,K,M,U,Y} |
| T5 | {C,E,I,K,O,O} |

| Trans ID | Items | Ordered itemset |
|---|---|---|
| T1 | {E,K,M,N,O,Y} | {K,E,M,O,Y} |
| T2 | {D,E,K,N,O,Y} | {K,E,O,Y} |
| T3 | {A,E,K,M} | {K,E,M} |
| T4 | {C,K,M,U,Y} | {K,M,Y} |
| T5 | {C,E,I,K,O,O} | {K,E,O} |

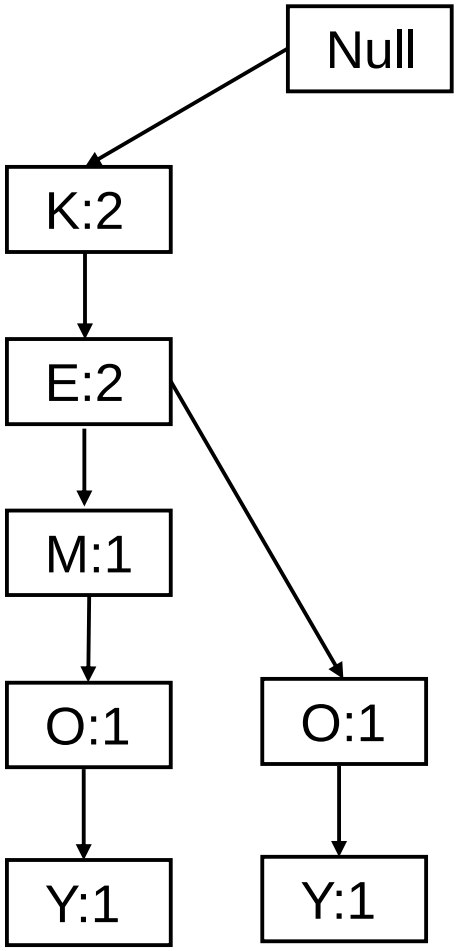
- **minsup=3**
- **Frequent item ={K:5, E:4, M:3, O:3, Y:3}**

- **Sort frequent items in descending order**

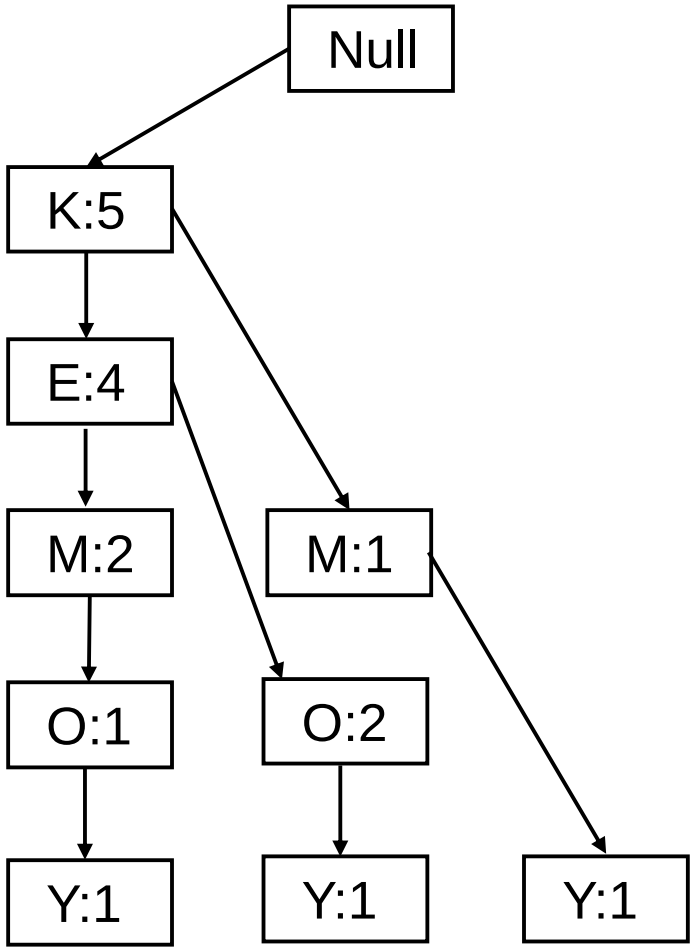
# FP-Tree

| Trans ID | Items | Ordered itemset |
|---|---|---|
| T1 | {E,K,M,N,O,Y} | {K,E,M,O,Y} |
| T2 | {D,E,K,N,O,Y} | {K,E,O,Y} |
| T3 | {A,E,K,M} | {K,E,M} |
| T4 | {C,K,M,U,Y} | {K,M,Y} |
| T5 | {C,E,I,K,O,O} | {K,E,O} |

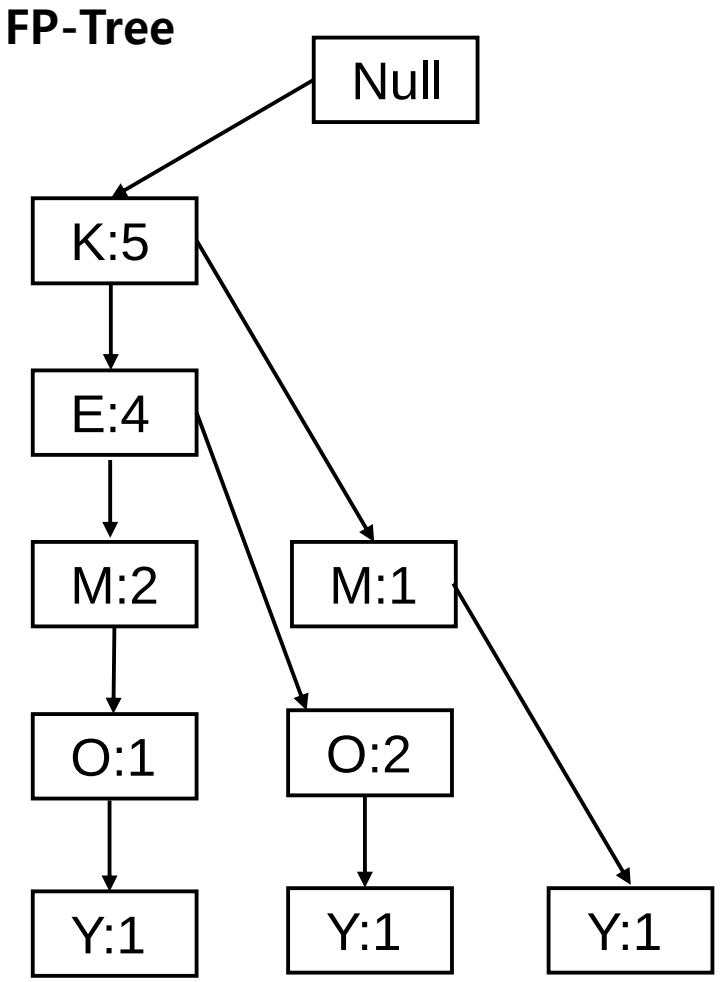
**After T1 & T2**




**After T1-T5**



# FP-Growth




- Now, for each frequent item in {K,E,M,O,Y}, the Conditional Pattern Base is computed from FP-Tree
- Conditional Pattern Base is labels of all the paths from root to the associated node in FP-Tree

| Items | Conditional Pattern Base |
|---|---|
| Y | {{K,E,M,O:1}, {K,E,O:1}, {K,M:1}} |
| O | {{K,E,M:1}, {K,E:2}} |
| M | {{K,E:2}, {K:1}} |
| E | {K:4} |
| K | |

# FP-Growth

| Items | Conditional Pattern base | Conditional FP tree |
|---|---|---|
| Y | {{K,E,M,O:1}, {K,E,O:1}, {K,M:1}} | {K:3} |
| O | {{K,E,M:1}, {K,E:2}} | {K,E:3} |
| M | {{K,E:2}, {K:1}} | {K:3} |
| E | {K:4} | {K:4} |
| K | | |

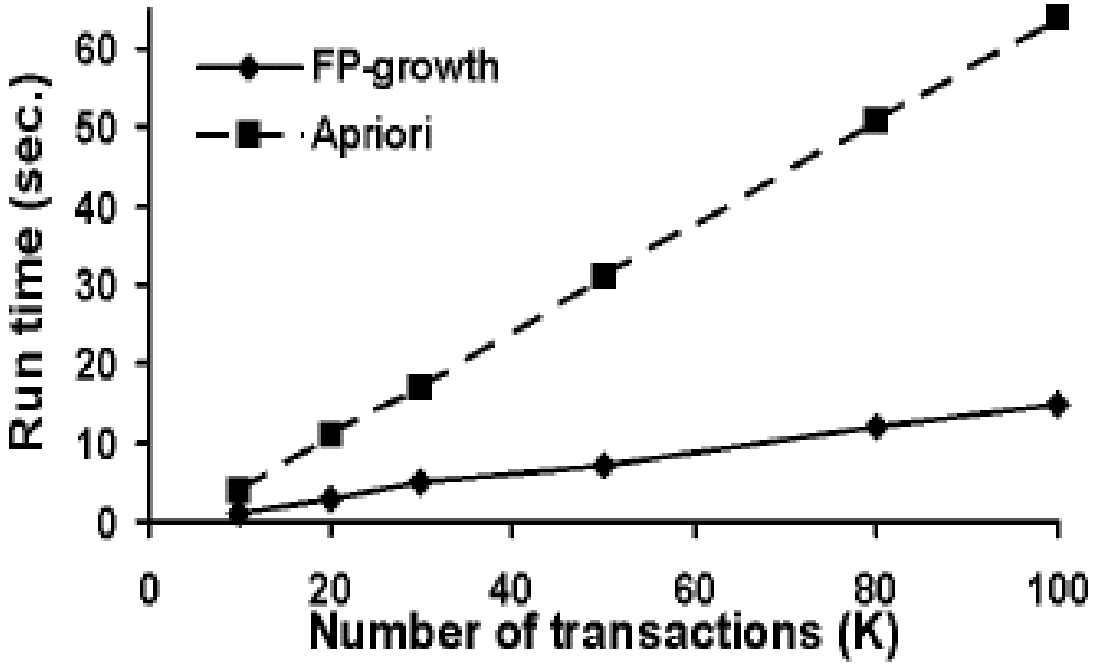
- Now compute Conditional FP Tree
- It is done by taking the set of elements which is common in all the paths in the Conditional Pattern Base
- Calculate it's support count by summing the support counts of all the paths in the Conditional Pattern Base.

| Items | Conditional FP tree | Frequent Pattern Generated |
|---|---|---|
| Y | {K:3} | {<K,Y:3>} |
| O | {K,E:3} | {<K,O:3>, <E,O:3>, <E,K,O:3>} |
| M | {K:3} | {<K,M:3>} |
| E | {K:4} | {<E,K:3>} |
| K | | |

- From the Conditional FP tree, the Frequent Patterns are generated by pairing the items of the Conditional FP Tree with the corresponding item.

# Performance Evaluation: FP-Tree vs. Apriori (Cont.)

- Scalability with DB size.



# FP-Growth

- An important method for solving important DM tasks
- Fast
- Compact
- Scalable
- Performance gets very important as databases are getting huge
- FP-Growth  has both performance and scalability

# FP-Growth in Python

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, fpgrowth

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
     ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],   ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
     ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],   ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

fpgrowth(df, min_support=0.6, use_colnames=True)
```

| | support | Itemsets |
|---|---|---|
| 0 | 1.0 | (Kidney Beans) |
| 1 | 0.8 | (Eggs) |
| 2 | 0.6 | (Yogurt) |
| 3 | 0.6 | (Onion) |
| 4 | 0.6 | (Milk) |
| 5 | 0.8 | (Kidney Beans, Eggs) |
| 6 | 0.6 | (Kidney Beans, Yogurt) |
| 7 | 0.6 | (Onion, Eggs) |
| 8 | 0.6 | (Onion, Kidney Beans) |
| 9 | 0.6 | (Onion, Eggs, Kidney Beans) |
| 10 | 0.6 | (Kidney Beans, Milk) |