COURSE NAME:
DATA WAREHOUSING & DATA MINING

LECTURE 15
TOPICS TO BE COVERED:

- Apriori Algorithm
- FP Growth

THE APRIORI ALGORITHM

- Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k+1)-itemsets
- Initially, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support.
- The resulting set is denoted L_1.Next, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.

THE APRIORI PROPERTY

- To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, presented below, is used to reduce the search space.
- Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*
- TheApriori property is based on the following observation.
- By definition, if an itemset *I does not satisfy the minimum support threshold, min sup, then I is not frequent; that* is, *$P(I) <$ min sup. If an item A is added to the itemset I, then the resulting itemset (i.e., $I \cup A$) cannot occur more frequently than I. Therefore, $I \cup A$ is not frequent either; that* is, *$P(I \cup A) <$ min sup.*

THE APRIORI PROPERTY

- *"How is the Apriori property used in the algorithm?" To understand this, let us look at* how L_{k-1} *is used to find* L_k *for k>=2. A two-step process is followed, consisting of join* and prune actions.
- **1. The join step:** To find L_k, *a set of candidate k-itemsets is generated by joining* L_{k-1} with itself.
- **2. The prune step:** C_k *is a superset of* L_k, *that is, its members may or may not be frequent,* but all of the frequent *k-itemsets are included in* C_k.

THE APRIORI ALGORITHM—AN EXAMPLE

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Example

Transactional data for an *AllElectronics* branch.

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Example

Scan D for count of each candidate →

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Compare candidate support count with minimum support count →

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Generate C_2 candidates from L_1 →

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

Scan D for count of each candidate →

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

Compare candidate support count with minimum support count →

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Generate C_3 candidates from L_2 →

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

Scan D for count of each candidate →

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Compare candidate support count with minimum support count →

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

: Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

THE APRIORI ALGORITHM

- Pseudo-code:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items};
  for (k = 1; L_k !=$\emptyset$; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
  increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

IMPORTANT DETAILS OF APRIORI

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

HOW TO GENERATE CANDIDATES?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into $\boldsymbol{C_k}$

  select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

  from $\boldsymbol{L_{k-1}\ p, L_{k-1}\ q}$

  where $\boldsymbol{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$
- Step 2: pruning

  forall ***itemsets c in*** $\boldsymbol{C_k}$ do

  forall ***(k-1)-subsets s of c*** do

  if *(s is not in* L_{k-1}*)* **then delete** *c* **from** C_k

FREQUENT PATTERN GROWTH

It can Suffer from two nontrivial cost:

- *It may need to generate a huge number of candidate sets.*
- *It may need to repeatedly scan the database and check a large set of candidates by pattern matching.*
- An interesting method in this attempt is called **frequent-pattern growth**, or simply FP-growth, which adopts a *divide-and-conquer strategy as follows. First, it* compresses the database representing frequent items into a frequent-pattern tree, or FP-tree, which retains the itemset association information. It then divides the compressed database into a set of *conditional databases (a special kind of projected database), each* associated with one frequent item or “pattern fragment,” and mines each such database separately.

Frequent Pattern Growth

- Frequent pattern growth (FP-growth) is a method of mining frequent itemsets without candidate generation. It constructs a highly compact data structure (an *FP-tree)* to compress the original transaction database. Rather than employing the generate and test strategy of Apriori-like methods, it focuses on frequent pattern (fragment) growth, which avoids costly candidate generation, resulting in greater efficiency.

Example

Transactional data for an *AllElectronics* branch.

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

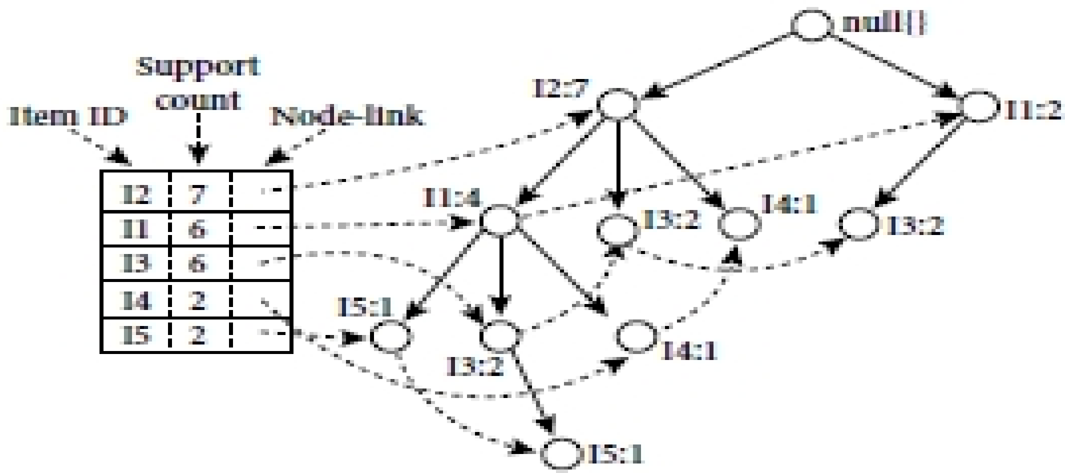
EXAMPLE

- FP-growth (finding frequent itemsets without candidate generation). The first scan of the database is the same as Apriori, which derives the set of frequent items (1-itemsets) and their support counts (frequencies). Let the minimum support count be 2. The set of frequent items is sorted in the order of descending support count. This resulting set or *list is denoted L. Thus, we have L ={{I2: 7}, {I1: 6}, {I3: 6},* {I4: 2}, {I5: 2}}.

Example

- An FP-tree is then constructed as follows. First, create the root of the tree, labeled with “null.” Scan database *D a second time.*
- The items in each transaction are processed in *L order (i.e., sorted according to descending support count), and a branch is created for* each transaction.
- For example, the scan of the first transaction, “T100: I1, I2, I5,” which contains three items (I2, I1, I5 in *L order), leads to the construction of the first branch of* the tree with three nodes, (I2: 1), (I1:1), and (I5: 1), where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I1. The second transaction, T200, contains the items I2 and I4 in *L order, which would result in a branch where I2 is linked to the root* and I4 is linked to I2. However, this branch would share a common prefix, I2, with the existing path for T100. Therefore, we instead increment the count of the I2 node by 1, and create a new node, (I4: 1),which is linked as a child of (I2: 2). In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

Example

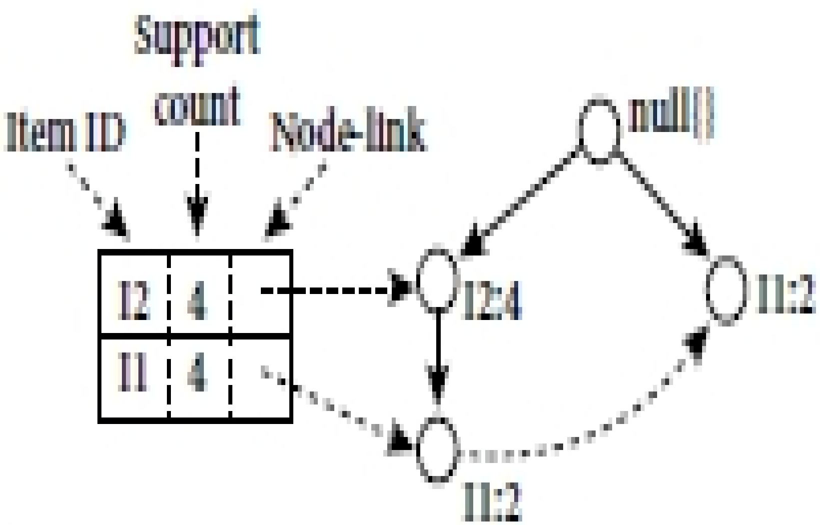


An FP-tree registers compressed, frequent pattern information.

Mining the FP-tree by creating conditional (sub-)pattern bases.

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

Example



The conditional FP-tree associated with the conditional node I3.

MINING FREQUENT ITEMSETS USING VERTICAL DATA FORMAT

- Mining frequent itemsets using vertical data format (ECLAT = Equivalence Class Transformation) is a method that transforms a given data set of transactions in the horizontal data format of *TID-itemset into* the vertical data format of *item-TID set. It mines the transformed data set by TID set* intersections based on the Apriori property and additional optimization techniques, such as *diffset.*

Example

Transactional data for an *AllElectronics* branch.

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Example

itemset	TID_set
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

The vertical data format of the transaction data set *D of given table*

Example

The 2-itemsets in vertical data format.

itemset	TID_set
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

The 3-itemsets in vertical data format.

itemset	TID_set
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

CHALLENGES OF FREQUENT PATTERN MINING

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

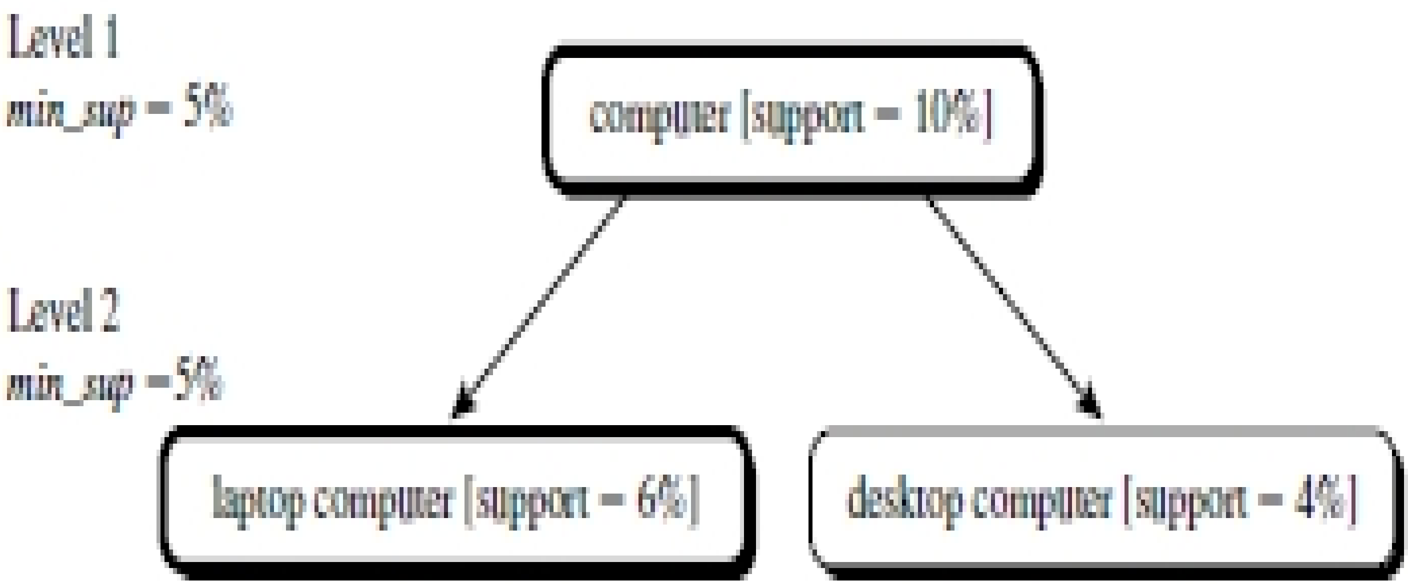
MINING VARIOUS KINDS OF ASSOCIATION RULES

- ***Multilevel association rules*** *involve concepts at different levels of abstraction.*
- ***Multidimensional association rules*** *involve more than one dimension or predicate* (e.g., rules relating what a customer *buys as well as the customer's age.)*
- ***Quantitative association rules*** *involve numeric attributes that have an implicit ordering among values* (e.g., *age).*

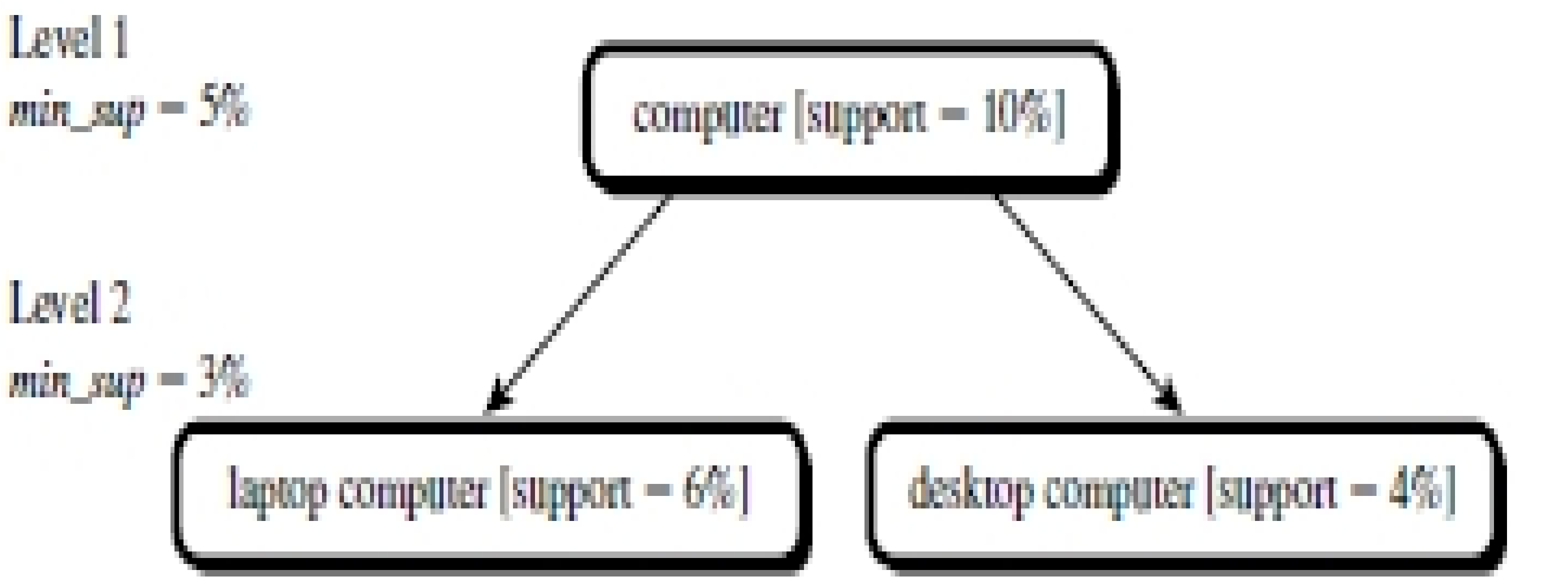
MULTILEVEL ASSOCIATION RULES.

- Association rules generated from mining data at multiple levels of abstraction are called **multiple-level or multilevel association rules.**
- Multilevel association rules can be mined efficiently using concept hierarchies under a support-confidence framework.

- Using uniform minimum support for all levels (referred to as uniform support): The same minimum support threshold is used when mining at each level of abstraction. For example, in Figure Next Slide, a minimum support threshold of 5% is used throughout (e.g., for mining from *“computer” down to “laptop computer”).* Both *“computer” and “laptop computer” are found to be frequent, while “desktop computer” is not.*
- When a uniform minimum support threshold is used, the search procedure is simplified. The method is also simple in that users are required to specify only one minimum support threshold.
- AnApriori-like optimization technique can be adopted, based on the knowledge that an ancestor is a superset of its descendants: The search avoids examining item sets containing any item whose ancestors do not have minimum support.



Multilevel mining with uniform support.



Multilevel mining with reduced support.

- If the threshold is set too low, it may generate many uninteresting associations occurring at high abstraction levels. This provides the motivation for the following approach.
- **Using reduced minimum support at lower levels (referred to as reduced support):** Each level of abstraction has its own minimum support threshold. The deeper the level of abstraction, the smaller the corresponding threshold is. For example, in Figure 5.12, the minimum support thresholds for levels 1 and 2 are 5% and 3%, respectively. In this way, *"computer," "laptop computer," and "desktop computer" are* all considered frequent.
- **Using itemor group-based minimum support (referred to as group-based support):** Because users or experts often have insight as to which groups are more important than others, it is sometimes more desirable to set up user-specific, item, or group based minimal support thresholds when mining multilevel rules. For example, a user could set up the minimum support thresholds based on product price, or on items of interest, such as by setting particularly low support thresholds for *laptop computers* and *flash drives in order to pay particular attention to the association patterns containing* items in these categories.

MINING MULTIDIMENSIONAL ASSOCIATION RULES FROM RELATIONAL DATABASES AND DATA WAREHOUSES

- Association rules that imply a single predicate, that is, the predicate *buys. For instance, in mining our AllElectronics database, we may* discover the Boolean association rule.

$$buys(X, \text{"digital camera"}) \Rightarrow buys(X, \text{"HP printer"}).$$

- This Rule refer as a singledimensional or intradimensional association rule because it contains a single distinct predicate (e.g., *buys)with multiple occurrences (i.e., the predicate occursmore than once* within the rule).

- An association rules containing *multiple predicates, such as*

$$age(X, \text{"20...29"}) \wedge occupation(X, \text{"student"}) \Rightarrow buys(X, \text{"laptop"}).$$

- Association rules that involve two or more dimensions or predicates can be referred to as multidimensional association rules. *It* contains three predicates (*age, occupation,* and *buys), each of which occurs only once in the rule. Hence, we say that it has no* repeated predicates. Multidimensional association rules with no repeated predicates are called interdimensional association rules.
- We can also mine multidimensional association rules with repeated predicates, which contain multiple occurrences of some predicates. These rules are called hybrid-dimensional association rules. An example of such a rule is the following, where the predicate *buys is repeated:*

$$age(X, \text{"20...29"}) \wedge buys(X, \text{"laptop"}) \Rightarrow buys(X, \text{"HP printer"})$$

MINING QUANTITATIVE ASSOCIATION RULES

- Quantitative association rules are multidimensional association rules in which the numeric attributes are *dynamically discretized during the mining process so as to satisfy* some mining criteria, such as maximizing the confidence or compactness of the rules mined. In this section, we focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side of the rule and one categorical attribute on the right-hand si… is,

$$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$$

- where A_{quan1} *and* A_{quan2} *are tests on quantitative attribute intervals (where the intervals* are dynamically determined), and A_{cat} *tests a categorical attribute from the task-relevant* data. Such rules have been referred to as two-dimensional quantitative association rules, because they contain two quantitative dimensions. For instance, suppose you are curious about the association relationship between pairs of quantitative attributes, like customer age and income, and the type of television (such as *high-definition TV, i.e., HDTV) that* customers like to buy. An example of such a 2-D quantitative association rule is

$$age(X, \text{"30...39"}) \wedge income(X, \text{"42K...48K"}) \Rightarrow buys(X, \text{"HDTV"})$$