What Is Frequent Pattern Analysis?

- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
- Applications
  - Market Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

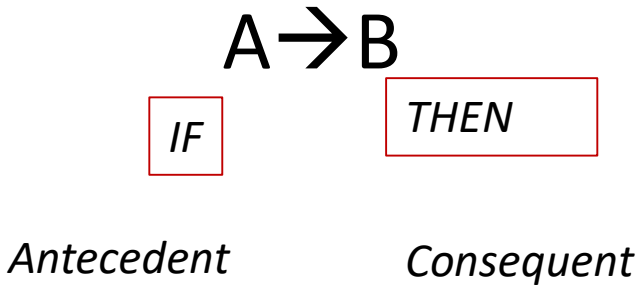
Frequent Itemsets

- A set of items is referred to as an itemset
- E.g., X = {milk, bread, cereal} is an itemset.
- An itemset that contains *k items is a k-itemset*.
- *The set {computer, antivirus software} is a 2-itemset*.

Association Rule

- Association rule can be thought of as an IF -> THEN relationship.
- Consider the rule A -> B.
- If a customer buys a product A, then the chances of item B too being bought by the customer under the same Transaction ID is found out.
- A co-occurance pattern comes into picture.

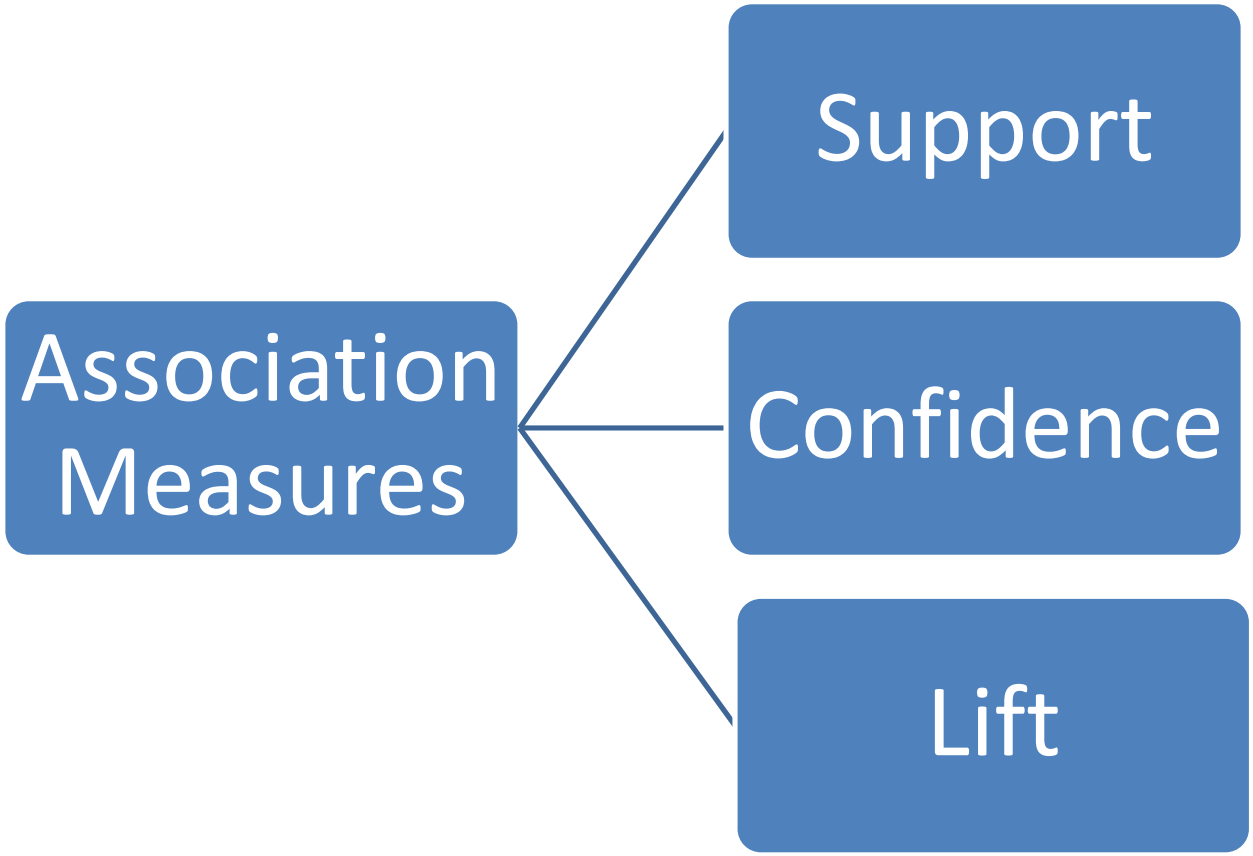
Association Rule



- IF contains an item or an item group that are typically found in the itemset.
- THEN contains items that comes along with an antecedent group.

{Bread, Butter}=> {Egg}

Association Rules



Association Rule Mining

$$Support = \frac{freq(A,B)}{N}$$

$$Confidence = \frac{freq(A,B)}{freq(A)}$$

$$Lift = \frac{Support}{Supp(A)\ x\ Supp(B)}$$

Support and Confidence

Example: An example is mined from the (some store) AllElectronics transactional database.

buys (X, “Computers”) ➔ buys (X, “software”) [Support = 1%, confidence = 50%]

- X represents customer
- Support = 1%, means that 1% of all the transactions under analysis showed that computer and software were purchased together.
- confidence = 50% , if a customer buys a computer there is a 50% chance that he/she will buy software as well.

Frequent Itemsets

- Find all the rules $X \rightarrow Y$ with minimum support and confidence
  - support, *s*, probability that a transaction contains $X \cup Y$
  - confidence, *c*, conditional probability that a transaction having X also contains *Y*

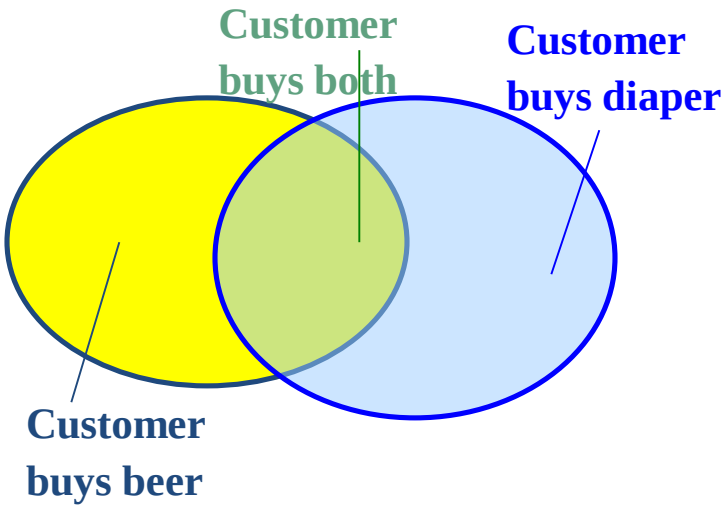
$$support(X => Y) = P(X \cup Y)$$

$$confidence(X => Y) = P(Y|X)$$

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Transaction-id	Items bought
10	A, B, D
20	A, C, D
30	A, D, E
40	B, E, F
50	B, C, D, E, F



Let $sup_{min} = 50\%$, $conf_{min} = 50\%$

Freq. Pat.: {*A:3, B:3, D:4, E:3, AD:3*}

Association rules:

A → *D* (60%, 100%)

Confidence= $\frac{3}{3}$ =100%

Support= $\frac{3}{5}$ = 60%

D → *A* (60%, 75%)

Confidence= $\frac{3}{4}$ =75%

Support= $\frac{3}{5}$ = 60%

Association rule mining

- Find all frequent itemsets**:** By definition, each of these itemsets will occur at least frequently as a predetermined minimum support count, *min_sup*.
- Generate strong association rules from the frequent itemsets**:** These rules must satisfy minimum support and minimum confidence.

Support and Confidence

- If the support of an itemset *I satisfies a pre-specified minimum support, then I is a* frequent itemset.
- The set of frequent *k*-itemsets is commonly denoted by L_k
- Rules that satisfy both a minimum support threshold (*min_sup*) *and a minimum confidence* threshold (*min_conf*) *are called strong*.

Closed Frequent Itemset and Maximal Frequent Itemset

- A long itemset will contain a combinatorial number of shorter, frequent sub-itemsets.
- A frequent itemset of length 100, such as {*a1, a2, : : : , a100}, contains* 100 frequent 1-itemsets: *a1, a2, : : : , a100,* frequent 2-itemsets: *(a1, a2), (a1, a3), : : : , (a99, a100), and so on.*
- *Itemset of length k, then* 2^k-1 *combinations are possible.*
- i.e; Too huge a number of itemsets for any computer to compute or store.
- Solution: *closed frequent itemset and maximal frequent itemset*

Closed Frequent Itemset

An itemset X is closed frequent itemset if X is *frequent* and there exists *no super-itemset* Y ⊃ X, *with the same support* as X

- An itemset X is considered a closed frequent itemset if:

  X is frequent: The itemset X appears frequently in the dataset (i.e., its support meets or exceeds a minimum threshold).

  No super-itemset Y with the same support: There does not exist any super-itemset Y (where X⊆Y) that has the same support as X.

- Implication: This means that X is a closed frequent itemset if it is frequent and there is no larger itemset that includes X and has the same frequency.
- Closed frequent itemsets help in reducing the number of itemsets we need to consider by eliminating redundant itemsets that do not provide any additional information.

Maximal Frequent Itemset

- An itemset X is a max-itemset if X is frequent and there exists no frequent super-itemset Y ⊃ X
- An itemset X is considered a max-itemset if:
  - X is frequent: The itemset X appears frequently in the dataset (i.e., its support meets or exceeds a minimum threshold).
  - No frequent super-itemset Y: There does not exist any super-itemset Y (where X⊆Y) that is also frequent.
- Implication: This means that X is a max-itemset if it is frequent and there are no larger itemsets that include X and are also frequent.
- Max-itemsets represent the largest sets of items that are frequent, and they help in understanding the boundaries of frequent itemsets in the dataset.

- Exercise. A transactional DB = {(a_1, …, a_{100}); (a_1, …, a_{50})}
  - Minimum support count threshold min_sup = 1.
- What is the set of closed itemset?
  - <a_1, …, a_{100}>: 1
  - < a_1, …, a_{50}>: 2
- What is the set of max-itemset?
  - <a_1, …, a_{100}>: 1
- Closed frequent itemsets contains complete information regarding the frequent itemsets
  - <a_2,a_{45}>: 2 since it is sub itemset of < a_1, …, a_{50}>: 2
  - <a_8,a_{55}>: 1 since it is sub itemset of < a_1, …, a_{100}>: 1
- From maximum itemset we can only assert that both itemsets <a_2,a_{45}> and <a_8,a_{55}> are frequent but we cannot assert their actual support counts

Efficient and Scalable Frequent Itemset Mining Methods- *Apriori Algorithm*

- The Apriori algorithm in data mining can help data analysts understand the underlying patterns in their data and help businesses handle their customers better.
- The Apriori algorithm is perfect for performing tasks such as market basket analysis.

Efficient and Scalable Frequent Itemset Mining Methods- *Apriori Algorithm*

- Algorithm uses prior knowledge of frequent itemset properties
- Apriori property: *If an itemset is frequent, then all of its subsets must also be frequent.*
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB and collecting those items that satisfy minimum support
  - Terminate when no frequent or candidate set can be generated

Apriori Algorithm

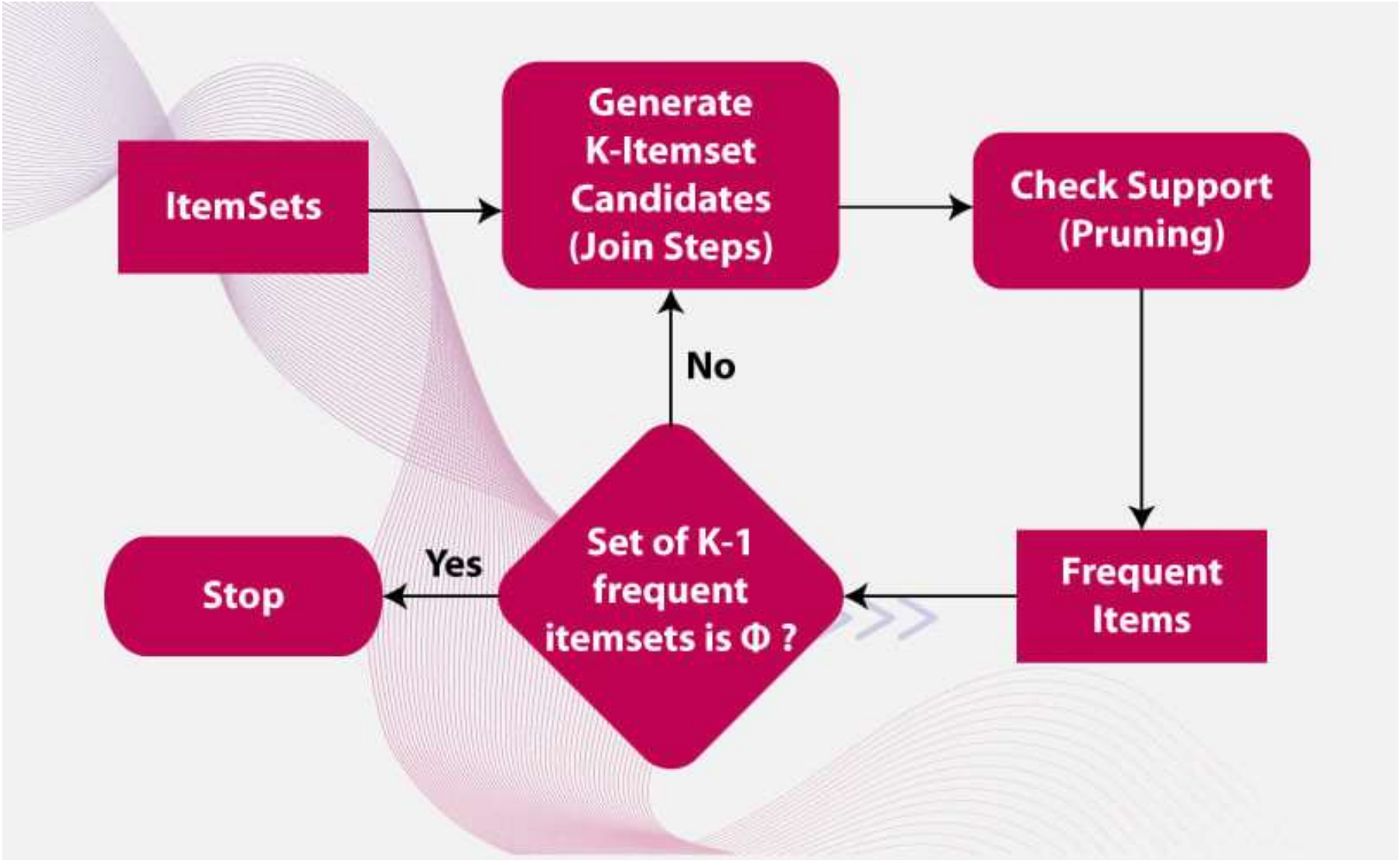
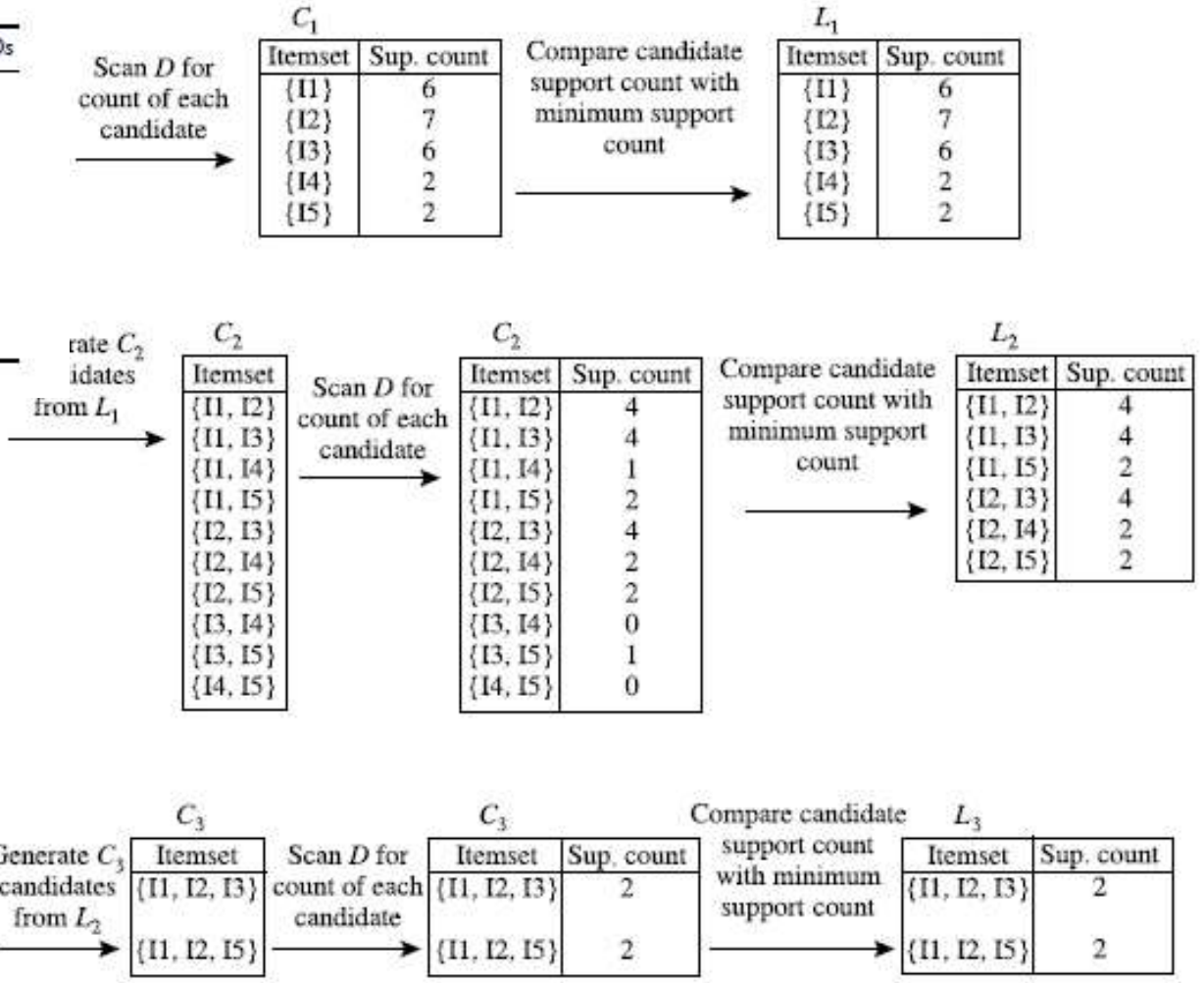


Table 6.1 Transactional Data for an *AllElectronics* Branch

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



e 6.2 Generation of the 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 !=∅; 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 L;

Important Details of Apriori

- How to generate candidates?
  - Step 1: Joining
  - Step 2: Pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: $L_3 * L_3$ ={*abc, abd, acd, ace, bcd*} *{*abc, abd, acd, ace, bcd*}
    ={abcd,acde}
  - 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

  for all ***itemsets c in*** $\boldsymbol{C_k}$ do

  for all ***(k-1)-subsets s of c*** do

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

Generating Association Rules from Frequent Itemsets

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support\_count(A \cup B)}{support\_count(A)}$$

- For each frequent itemset *l, generate all nonempty subsets of l.*
- For every nonempty subset *s of l, output the rule "s =>(l-s)" if* $\frac{support\ count(l)}{support\ count(s)} >= min\ conf$

  where min conf is the minimum confidence threshold.

Improving the Efficiency of Apriori

- Hash-based technique- when scanning for 1-itemset, generate all of the 2-itemsets for each transaction, hash them into the different *buckets of* a *hash table structure, and increase the corresponding bucket counts*

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y)) \bmod 7$

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Improving the Efficiency of Apriori

Transaction reduction

- A transaction that does not contain any frequent *k-itemsets cannot contain any* frequent (*k+1)-itemsets.*
- Such a transaction can be marked or removed from further consideration because subsequent database scans for j-itemsets, where j > k, will not need to consider such a transaction.
- Adv: Reducing the number of transactions scanned in future iterations

Partitioning

- Requires just two database scans to mine the frequent itemsets.
- It consists of two phases.

Phase I:

- The algorithm divides the transactions of D into n nonoverlapping partitions.
- If the minimum relative support threshold for transactions in D is min_sup, then the minimum support count for a partition is :

  min_sup X the number of transactions in that partition
- For each partition, all the local frequent itemsets (i.e., the itemsets frequent within the partition) are found.

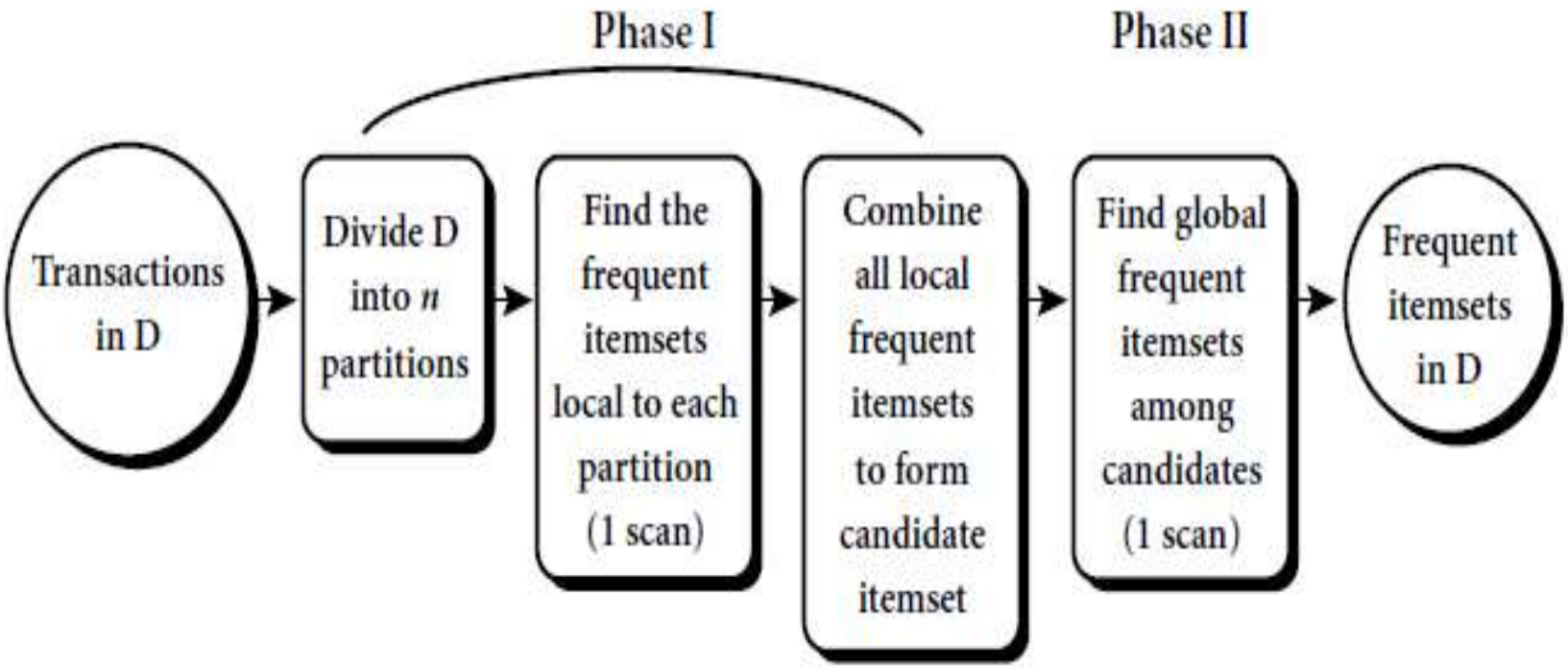
- A local frequent itemset may or may not be frequent with respect to the entire database, *D*.
- However, *any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions*.
- Therefore, all local frequent itemsets are candidate itemsets with respect to *D*.
- The collection of frequent itemsets from all partitions forms the *global candidate itemsets* with respect to *D*.

Phase II:

- A second scan of D is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets.
- Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Partitioning

- Requires just two database scans to mine the frequent itemsets



Sampling

- Pick a random sample *S of the given data D, and then search for frequent* itemsets in *S instead of D.*
- *Result:* we trade off some degree of accuracy against efficiency.
- *The S sample size is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall.*
- *We are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets.*
- *Solution:* Use a lower support threshold than minimum support to find the frequent itemsets local to S (denoted L^S).

Dynamic itemset counting

- A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points.
- A new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.
- The technique uses the count-so-far as the lower bound of the actual count.
- If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collection and can be used to generate longer candidates.
- This leads to fewer database scans than with Apriori for finding all the frequent itemsets.

Mining Frequent Patterns Without Candidate Generation

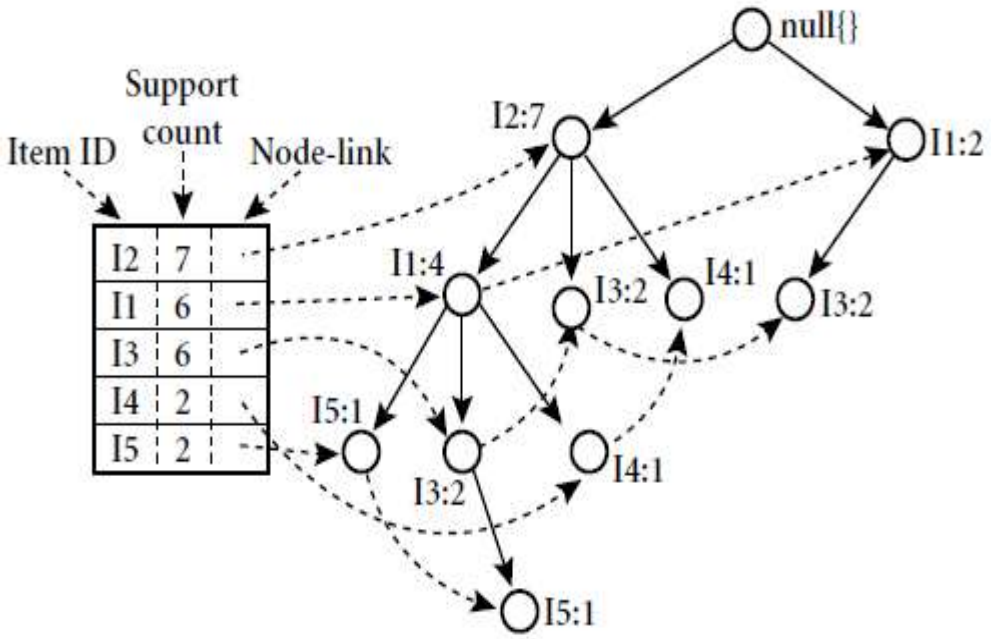
- Problems with Apriori candidate generate and test method:

  - *It may still need to generate a huge number of candidate sets.* For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.
  - *It may need to repeatedly scan the whole database and check a large set of candidates by pattern matching.* It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

Mining Frequent Patterns Without Candidate Generation

- Frequent-pattern growth, or FP-growth
  - *Divide-and-conquer strategy*
  - Compresses the database representing frequent items into a frequent-pattern tree, or FP-tree which retains the iremset association information.
  - Divides the compressed database into a set of *conditional databases, each* associated with one frequent item or "pattern fragment," and mines each such database separately.

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



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}