Association Rule Mining - From Market Baskets to Interesting Patterns

Duration: 1 Hour

- Prerequisites:
- Basic probability, set theory
- understanding of data mining objectives

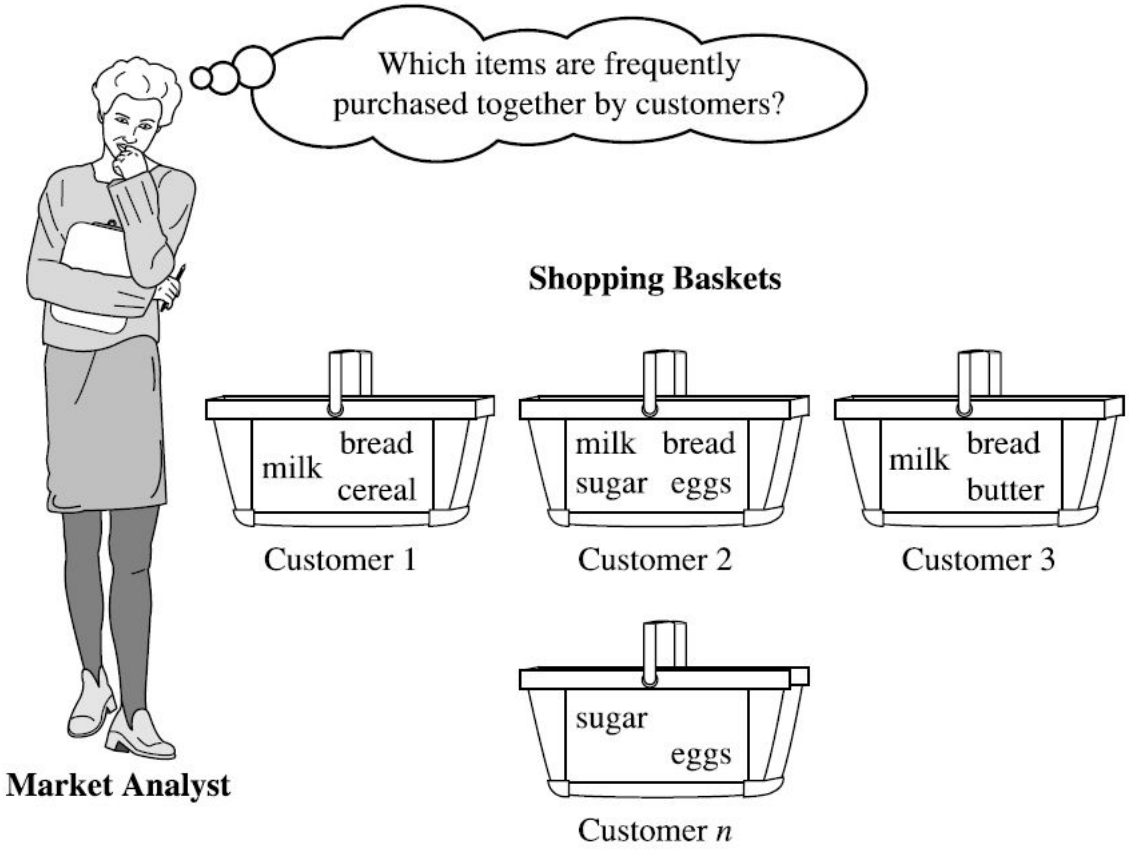


FIGURE 4.1

Market basket analysis.

A Motivating Example

In 1993, Walmart made a surprising discovery.

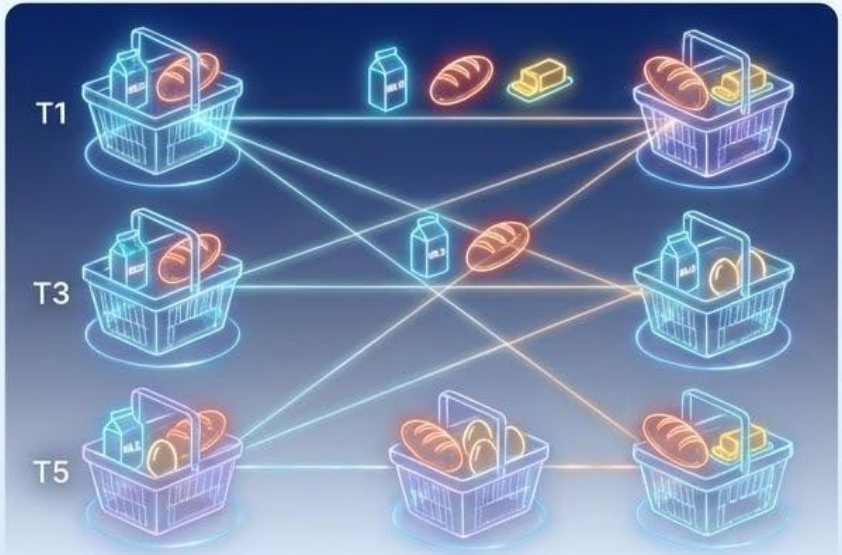
The Data & Insight:

- Data analysis revealed that beer and diapers were frequently purchased together on Friday evenings.
- The Insight: Young fathers buying diapers for the baby also bought beer for the weekend.
- This led to strategic product placement and targeted promotions.

This is the foundation of Market Basket Analysis.

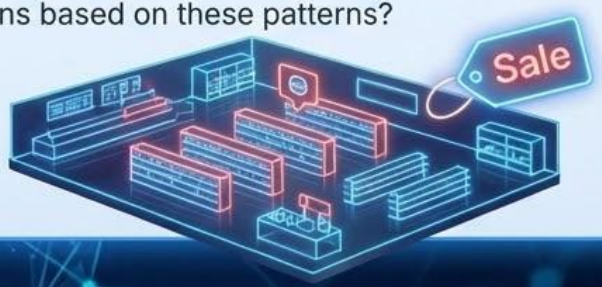
Market Basket Data

Transaction Data



Business Questions

- **Co-occurrence:** What items are frequently bought together?
- **Prediction:** If someone buys *Item X*, what else are they likely to buy?
- **Strategy:** How can we optimize store layout and promotions based on these patterns?



Transaction ID	Items Purchased
T1	{Milk, Bread, Butter}
T2	{Bread, Butter}
T3	{Milk, Bread}
T4	{Milk, Eggs}
T5	{Bread, Eggs, Butter}

Key Definitions: Itemsets & Support

Itemset 

A collection of one or more items.

- {Milk}
- {Milk, Bread}
- {Bread, Butter, Eggs}

Support 

The fraction of transactions containing an itemset. It measures frequency.

Example Calculation:

Total Transactions = 5

Support({Milk}) = 3/5 = **0.6**
(Appears in T1, T3, T4)

Support({Milk, Bread}) = 2/5 = **0.4**
(Appears in T1, T3)

Frequent Itemsets

The Threshold Concept

An itemset is considered **"frequent"** if its support is greater than or equal to a user-defined minimum support threshold (min_sup).

Logic:

If min_sup = 0.4

And Support({🥛Milk, 🍞Bread}) = 0.4

✅ Then {Milk, Bread} is **Frequent** (0.4 ≥ 0.4)

Association Rules & Metrics

Rule Form: X → Y (If X is bought, then Y is bought)

1. Support (X → Y)

Probability that a transaction contains X and Y.


```
Support({Milk, Bread}) = 0.4
```

2. Confidence (X → Y)

Conditional probability: How often Y occurs given X.

```
Conf = Supp(X ∪ Y) / Supp(X)
0.4 / 0.6 = 0.667
```

"66.7% of milk purchases include bread"

X → Y

66.7% of milk purchases include bread

Advanced: Closed Itemsets

Definition

An itemset is **closed** if *none* of its immediate supersets has the same support count.

It implicitly stores the support of its subsets.

Examples

{Milk}

Support = 0.6

{Milk, Bread}

Support = 0.4

❌ **{Milk} is NOT closed** (Superset has different support)

{Bread}

Support = 0.8

{Bread, X}

Support < 0.8

✅ **{Bread} IS closed** (No superset equals 0.8)

Maximal Frequent Itemsets

A frequent itemset with **no frequent supersets**.

{A, B, C} is frequent...

{A, B, C, D} is **NOT** frequent...

Then {A, B, C} is Maximal.

Optimization: Why Use Them?

Closed Itemsets

Lossless Compression

We can derive the **exact support** of all subsets from **closed** itemsets.

~100 itemsets

Maximal Itemsets

Lossy Compression

We know the subsets are frequent, but we **lose** their specific **support** counts.

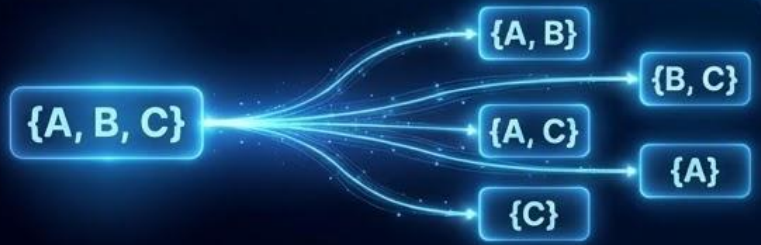
~20 itemsets

Example reduction: From 1,000 frequent itemsets to manageable numbers.

The Core Insight: Apriori Property

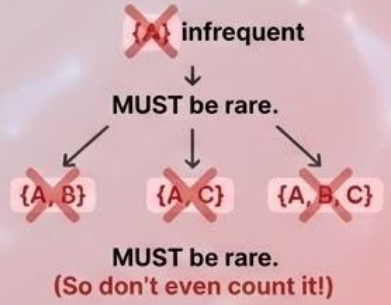
"All non-empty subsets of a frequent itemset must also be frequent."

This property is the key to efficiency. It allows us to "prune" the search space significantly.



The Pruning Corollary

If an itemset is **infrequent**, all its supersets are guaranteed to be infrequent.



Algorithm Steps

Pass 1:

Scan database to count 1-itemsets. Determine L_1.

L_1 (Frequent 1-itemsets)

Pass k (k ≥ 2): The K-Loop Logic

1. **Candidate Gen**: Create C_k from L_{k-1}.

2. **Pruning**: Remove candidates with infrequent subsets.

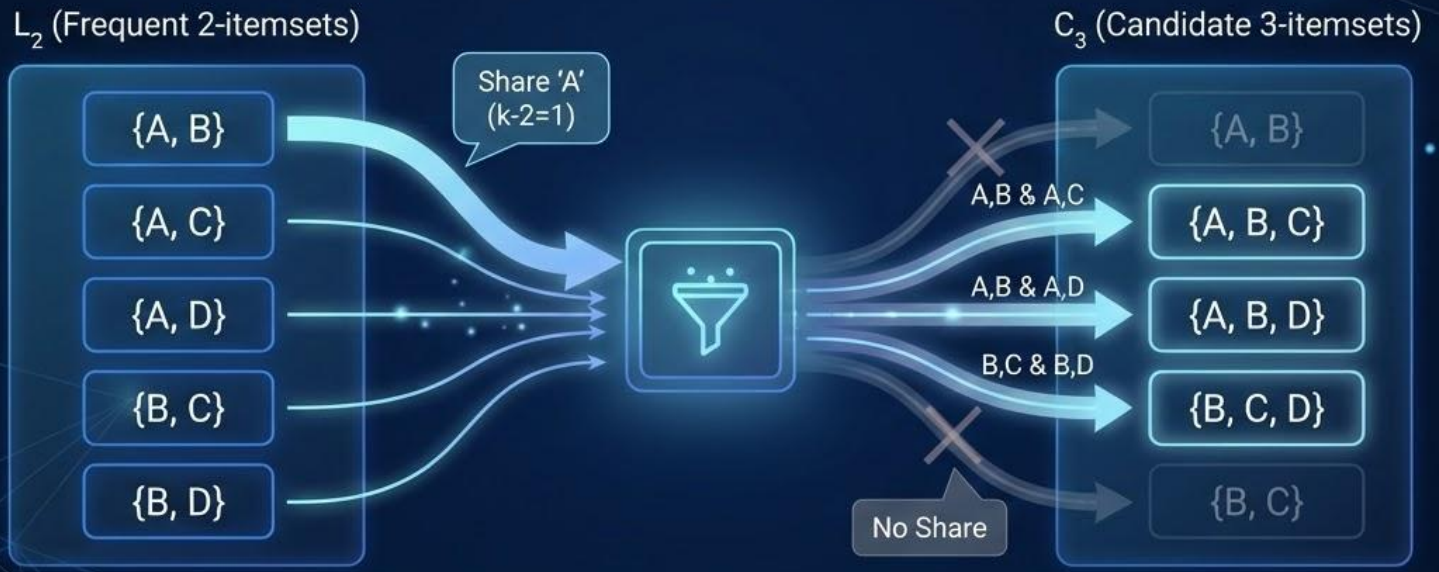
3. **Scan**: Count support for survivors.

4. **Filter**: Keep those ≥ **min_sup** (→ L_k).

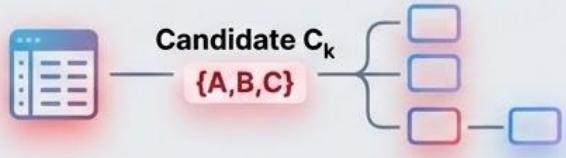
Repeat until no new frequent itemsets are found.

Step 1: Candidate Generation (Join)

To find frequent k-itemsets, we join frequent (k-1)-itemsets that share the first k-2 items.



Step 2: Pruning Step



Before scanning database, check generated generated candidates. Every subset of size (k-1) must be in L_{k-1}.

If **ANY** subset is missing, the candidate **canot** be frequent. Prune it.

Example: Checking {A, B, C}

Check all 2-subsets against L_2:

- ✓ Is {A, B} — ✅ Is in L_2? **Yes**
- ✓ Is {A, C} — ✅ Is in L_2? **Yes**
- ✗ Is {B, C} — ❓ Is in L_2? **NO**

If {B,C} is NOT in L_2, discard {A,B,C} immediately!

4.2.1 Apriori Algorithm: The Foundation

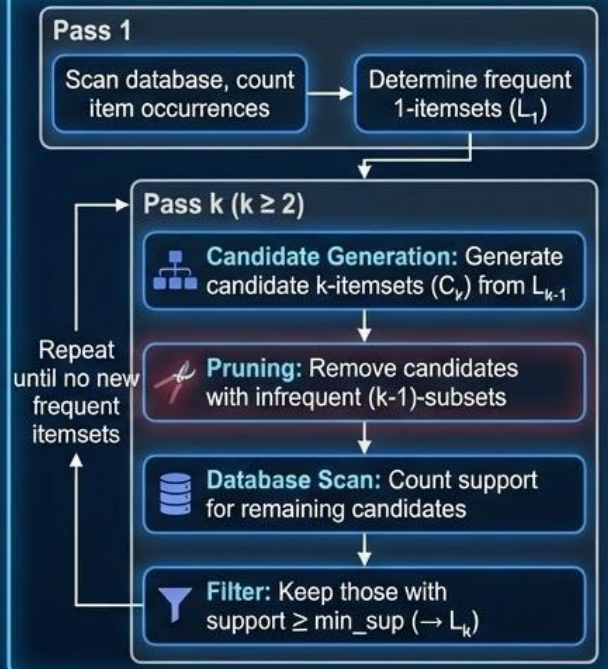
The Core Insight: Apriori Property

"All non-empty subsets of a frequent itemset must also be frequent"

- **Corollary:** If an itemset is infrequent, all its supersets are infrequent.

This enables pruning!

Apriori Algorithm Steps



Example Walkthrough

Database: T1: A,B,C
T2: A,C
T3: A,D
T4: B,E,F

min_sup = 2/4 = 0.5

Pass 1:
L_1 = {A:3, B:2, C:2} (D,E,F < 2)

Pass 2:
C_2 = {A,B}, {A,C}, {B,C}
Count: {A,B}:1, {A,C}:2, {B,C}:1
L_2 = {A,C:2}

Stop: No C_3 possible

Table 4.1 A transactional data set.

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

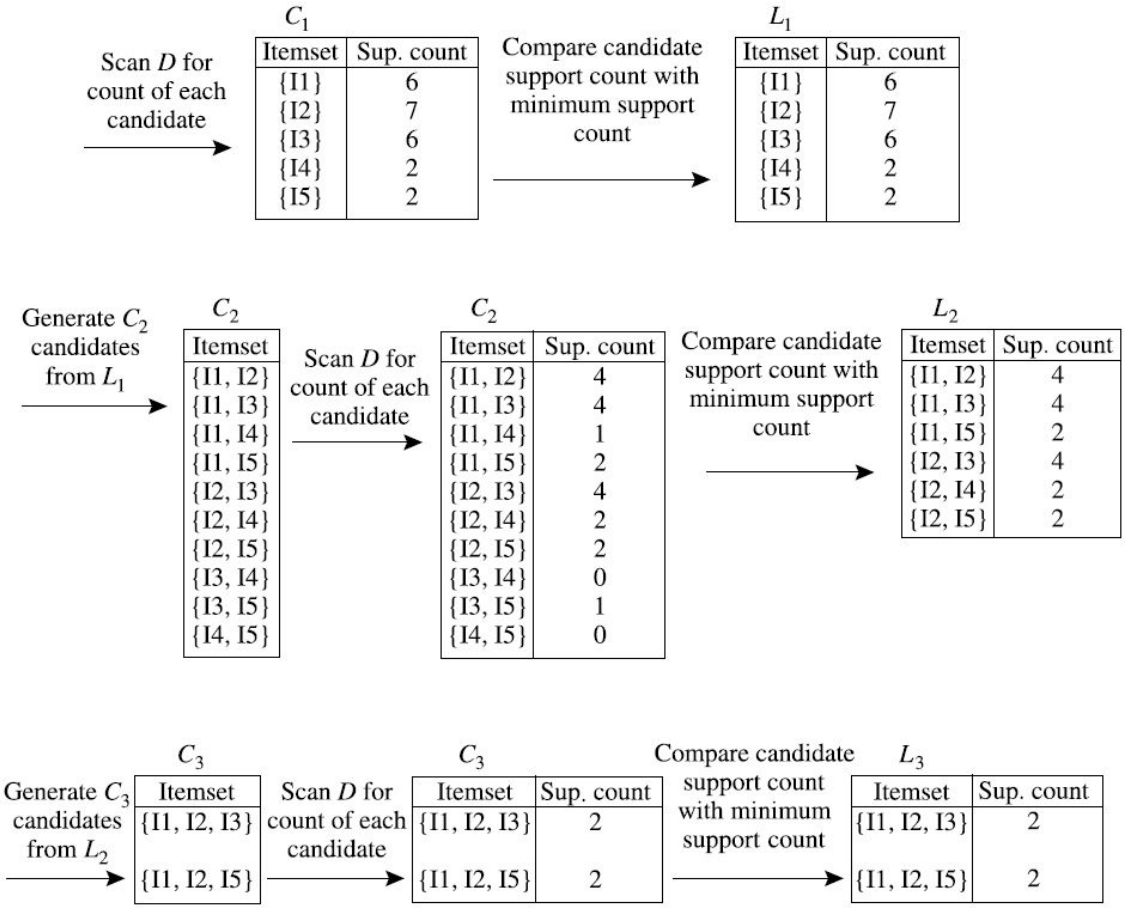


FIGURE 4.2

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

a. Join: $C_3 = L_2 \bowtie L_2 = \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$\bowtie \{\{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}\}$
$= \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$.

b. Prune using the Apriori property: *all nonempty subsets of a frequent itemset must also be frequent*. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L_2. Therefore, keep {I1, I2, I3} in C_3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

c. Therefore, $C_3 = \{\{I1, I2, I3\}, \{I1, I2, I5\}\}$ after pruning.

FIGURE 4.3

Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

Complexity & Limitations

The Bottleneck

Apriori is revolutionary but has a major flaw: **I/O Cost**.

- **Multiple Scans:** It requires one full database scan for *each* length k.
- **Huge Candidate Sets:** Generating candidates for size 2 (C_2) can be massive if L_1 is large (n items → n(n-1)/2 pairs).

Rule Generation Algorithm

Once we have a **Frequent Itemset (L)**, how do we create rules?

Step 1: Subsets
Generate all non-empty subsets of L.

Step 2: Form Rules
For every subset A, create the rule: **A → (L - A)**.

Step 3: Evaluate
Compute Confidence = Support(L) / Support(A).
Keep rule if **Confidence ≥ min_conf**.

Example: Breakdown

The Scenario

Let's assume we found a frequent itemset **L**.

```
L = {Milk, Bread, Butter}
Support(L) = 0.3
```

Step 1: Generate Subsets

We need to test rules based on these subsets:

- {Milk}
- {Bread}
- {Butter}
- {Milk, Bread}
- {Milk, Butter}
- {Bread, Butter}

Example: Rule Calculation

Let's evaluate one specific potential rule using a subset.

Parameters

- min_conf = 0.7
- Support({Milk, Bread}) = 0.4

The Rule & Formula

Rule: {Milk, Bread} → {Butter}

Formula:
Conf = Supp(L) / Supp({Milk, Bread})

Calculation:
Conf = 0.3 / 0.4 = 0.75

Evaluation & Conclusion

0.75 ≥ 0.7

Rule is Strong (Keep)

Optimization Strategy

We can optimize using **Confidence Anti-monotonicity**.

This allows us to prune rules without calculating every single combination.

The Principle

If rule **X → (Y - Z)** has **low confidence**, then all rules **X' → (Y - Z)** (where X' is a subset of X) also have **low confidence**.

Intuition: Shrinking the antecedent (X) usually increases the denominator (Support of X), thereby lowering confidence.

Example 4.4. Generating association rules. Let's try an example based on the transactional data shown before in Table 4.1. The data contain frequent itemset $X = \{I1, I2, I5\}$. What are the association rules that can be generated from X? The nonempty subsets of X are $\{I1, I2\}$, $\{I1, I5\}$, $\{I2, I5\}$, $\{I1\}$, $\{I2\}$, and $\{I5\}$. The resulting association rules are as shown below, each listed with its confidence:

$$\{I1, I2\} \Rightarrow I5, \quad confidence = 2/4 = 50\%$$
$$\{I1, I5\} \Rightarrow I2, \quad confidence = 2/2 = 100\%$$
$$\{I2, I5\} \Rightarrow I1, \quad confidence = 2/2 = 100\%$$
$$I1 \Rightarrow \{I2, I5\}, \quad confidence = 2/6 = 33\%$$
$$I2 \Rightarrow \{I1, I5\}, \quad confidence = 2/7 = 29\%$$
$$I5 \Rightarrow \{I1, I2\}, \quad confidence = 2/2 = 100\%$$

The Bottlenecks of Apriori

Multiple Scans

Requires a full I/O scan of the database for every single pass (k).

Huge Candidates

Generates an exponential number of candidate itemsets to check.

Tedious Counting

Support counting becomes computationally expensive and slow.

1. Hash-based Technique

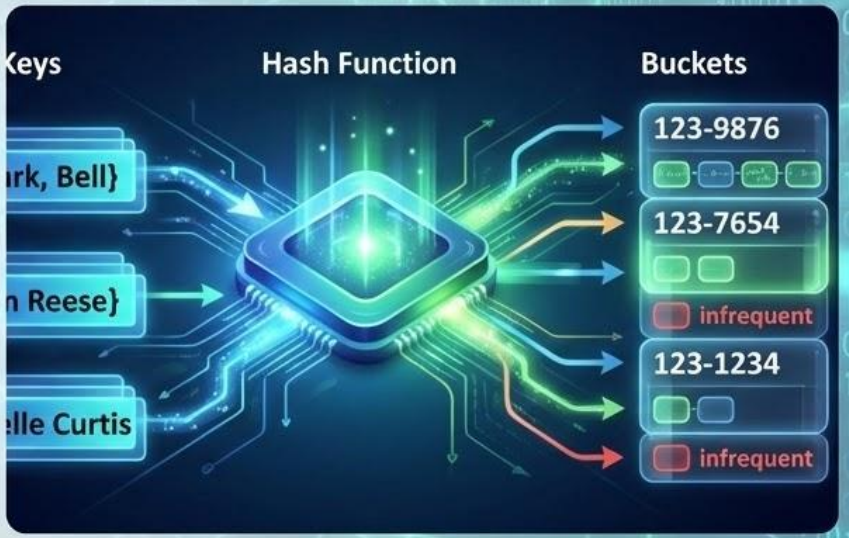
Reduce Candidate Pairs

Instead of counting every candidate, we can filter them early.

Filter Candidates Early.

- **Method:** Use a hash table to group itemsets into buckets.
- **Logic:** If the count of a bucket is less than the minimum support, *none* of the itemsets in that bucket can be frequent.

✅ **Result:** Only count colliding pairs in frequent buckets.



Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck|c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

FIGURE 4.4

Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

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

H_2

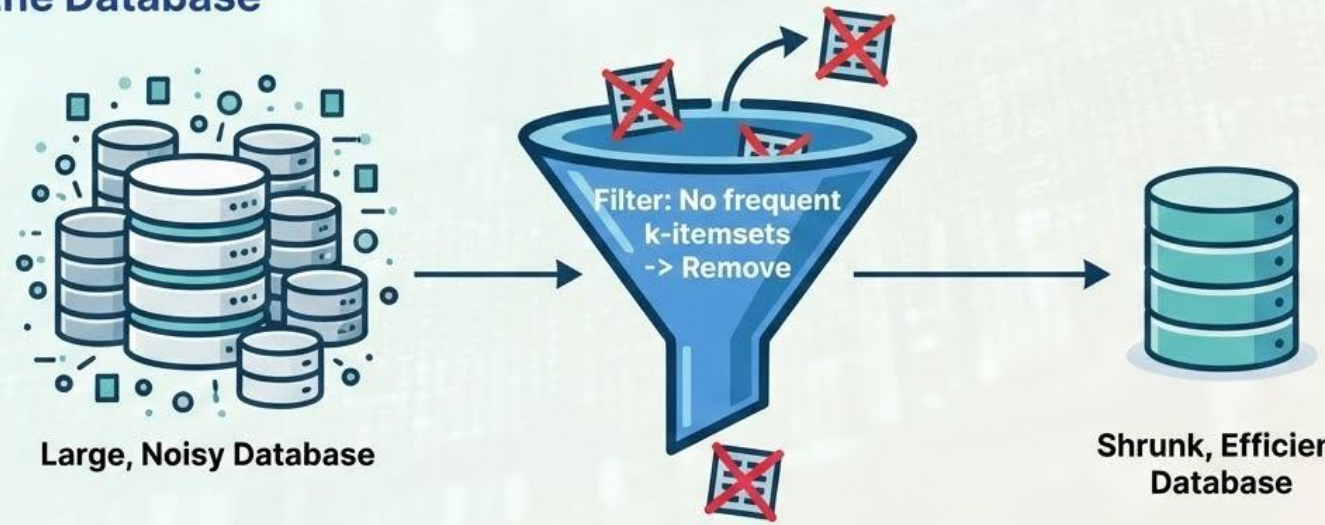
bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

FIGURE 4.5

Hash table, H_2, for candidate 2-itemsets. This hash table was generated by scanning Table 4.1's transactions while determining L_1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

2. Transaction Reduction

Shrink the Database



- **Action:** Identify and remove useless transactions in future scans.
- Reduces I/O cost.
- Makes subsequent passes faster.

Filter Out Noise

Transaction reduction (reducing the number of transactions scanned in future iterations). A transaction that does not contain any frequent k-itemsets cannot contain any frequent $(k+1)$-itemsets. Therefore 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.

3. Partitioning

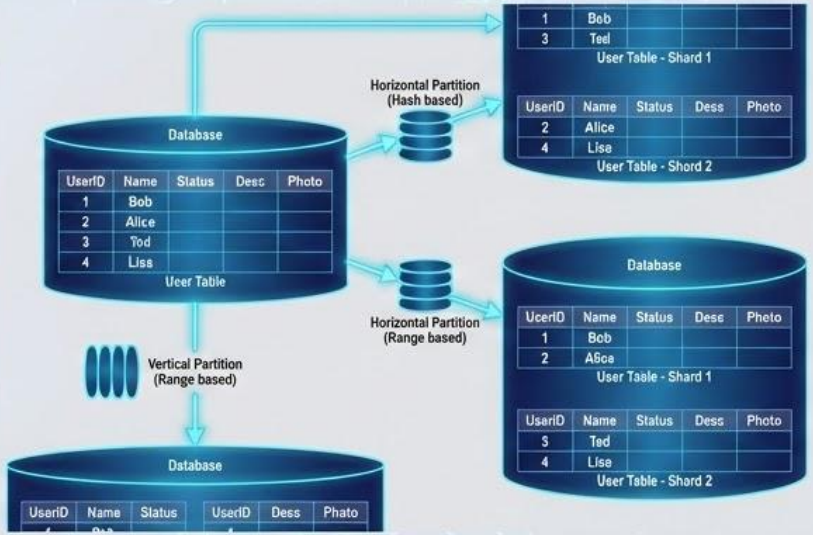
Divide and Conquer

Problem: The Database is too big for memory (RAM).

Solution: Break it down.

1. Divide database into partitions that fit in memory.
2. Find **Local Frequent Itemsets** in each partition.
3. Combine them to form the global candidate set.
4. Scan once more to verify global support.

.



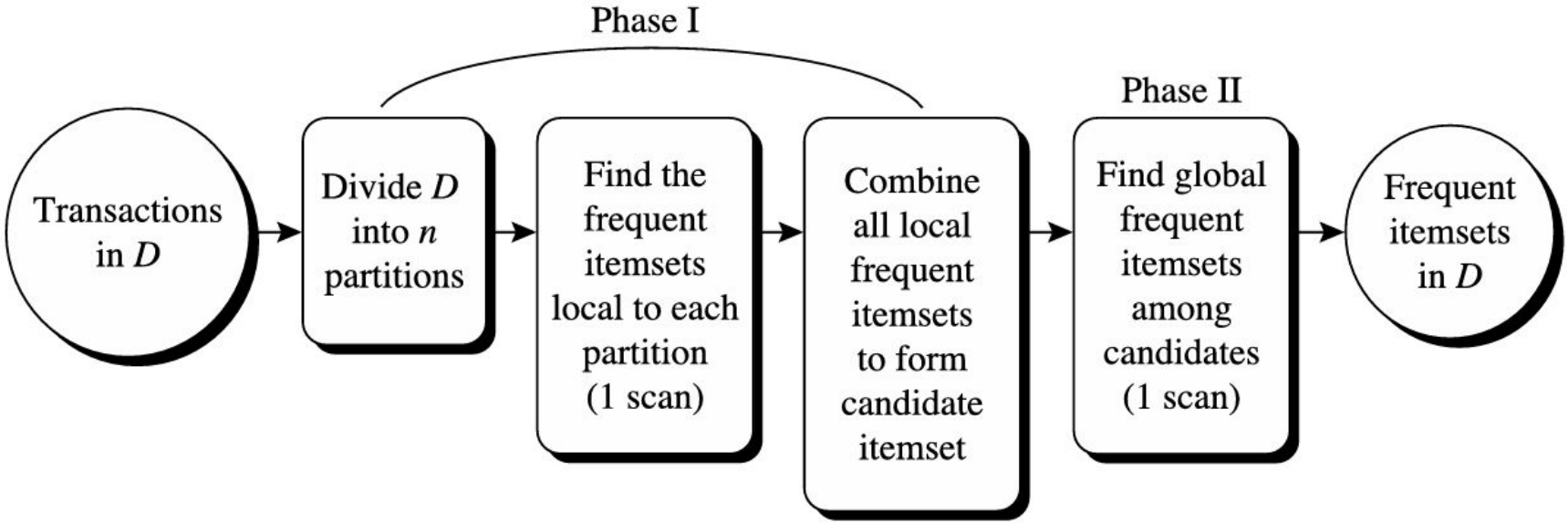


FIGURE 4.6

Mining by partitioning the data.

4. Sampling

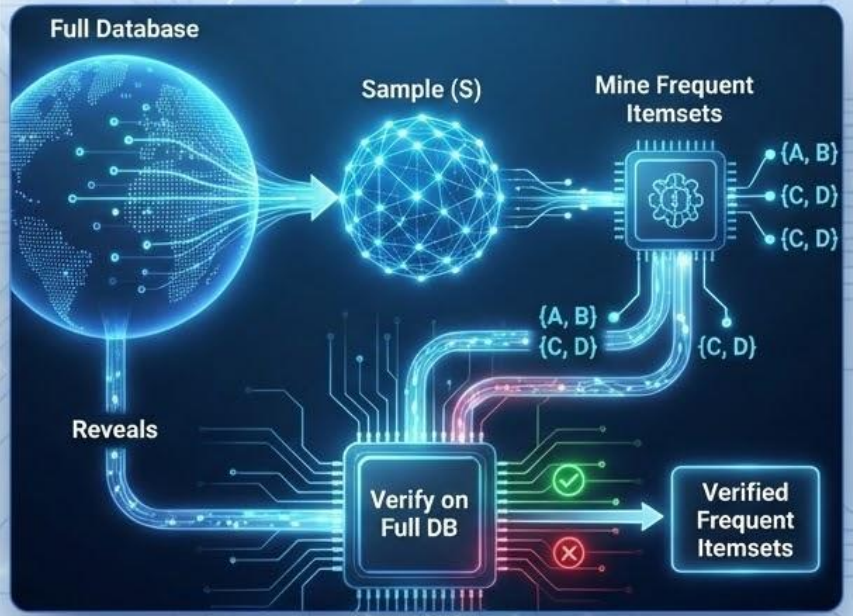
Trade-off: Speed vs. Accuracy

Instead of scanning the whole DB, pick a random sample.

Process: Mine frequent itemsets on the sample (S).

Verification: Verify the winners on the full database.

Risk: False Negatives. You might miss some itemsets that are frequent in the full DB but not the sample.



5. Dynamic Itemset Counting (DIC)

Start Early

Don't wait for a full scan to finish before starting the next phase. Start counting candidate support as soon as possible.

Eliminate Fast

If an itemset looks unpromising partway through, stop tracking it. Eliminate candidates early to save resources.

Pattern-Growth Approach: FP-Growth Algorithm

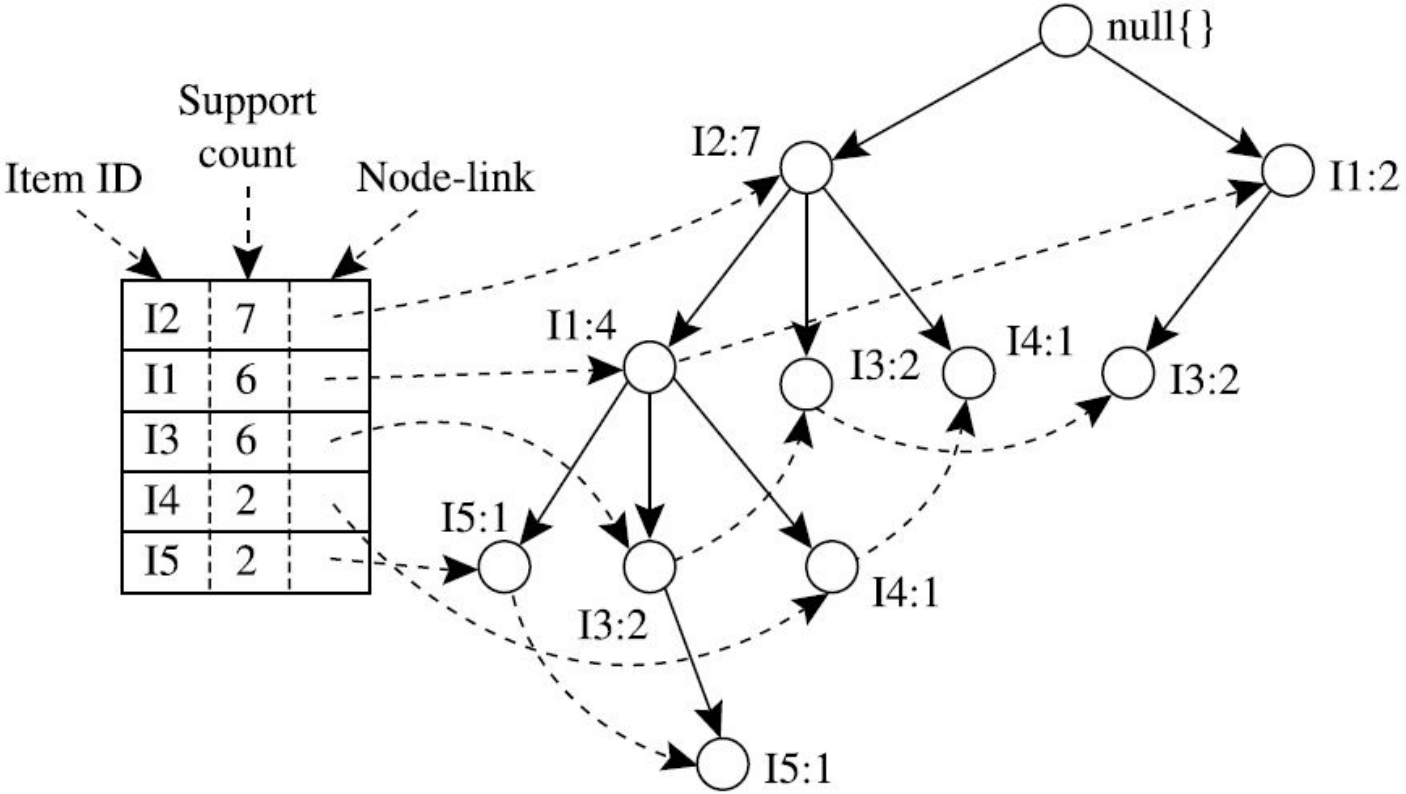


FIGURE 4.7

An FP-tree registers compressed frequent pattern information.

Table 4.2 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}

Algorithm: **FP_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.
Method:

1. The FP-tree is constructed in the following steps:
   a. Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   b. Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following.
   Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, α)
(1)  if Tree contains a single path P then
(2)      for each combination (denoted as β) of the nodes in the path P
(3)          generate pattern β ∪ α with support_count = minimum support count of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)      generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)      construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
(7)      if Tree_β ≠ ∅ then
(8)          call FP_growth(Tree_β, β); }
```

FIGURE 4.8

FP-growth algorithm for discovering frequent itemsets without candidate generation.

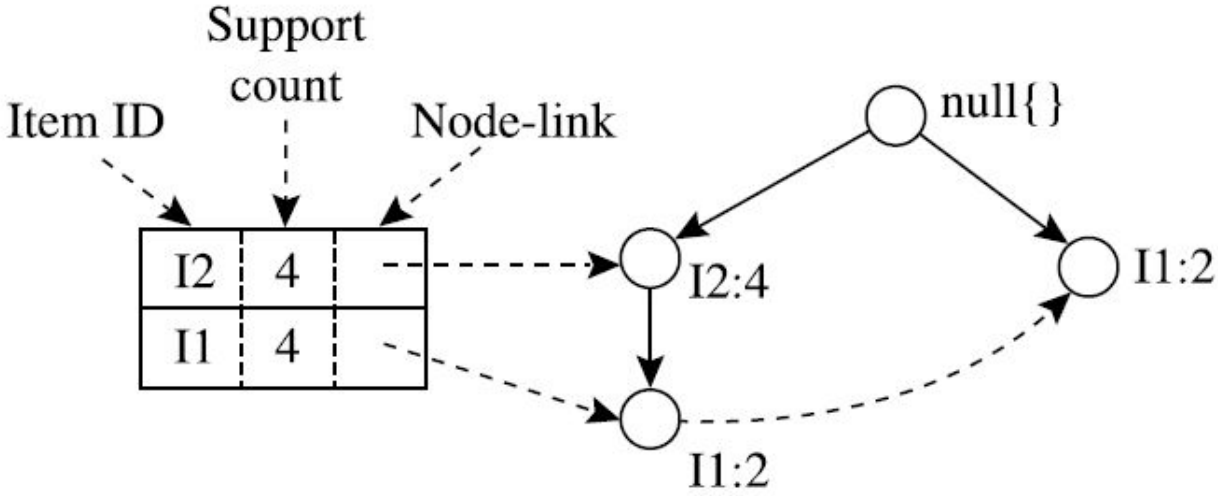


FIGURE 4.9

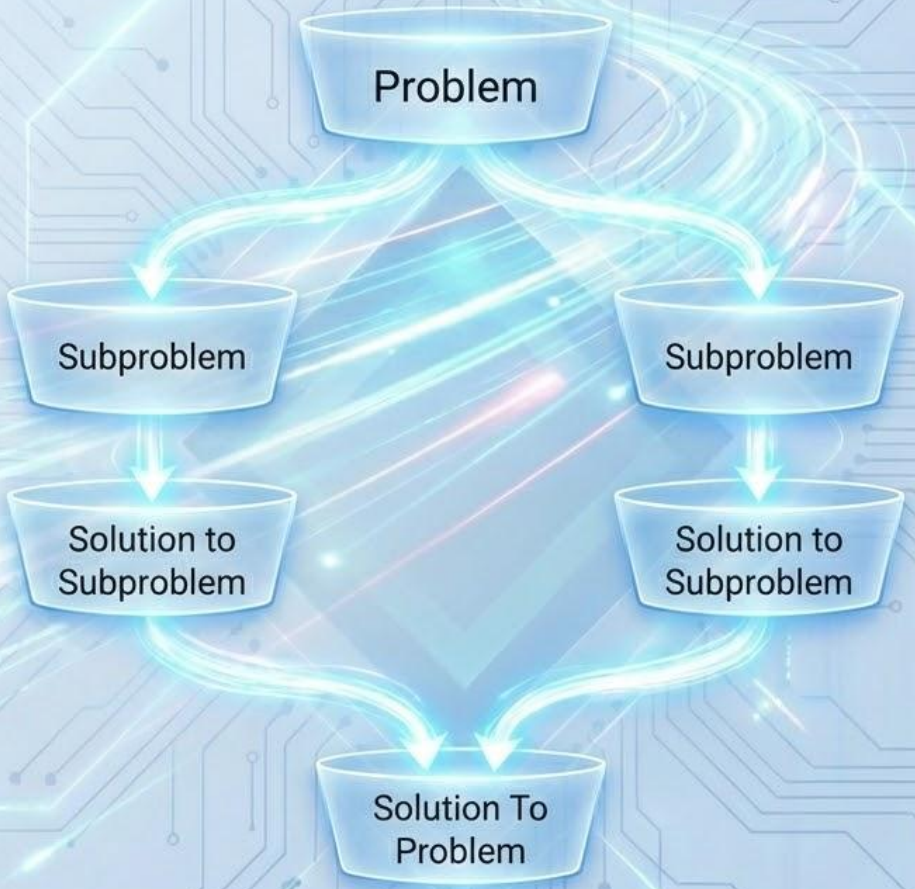
The conditional FP-tree associated with the conditional node I3.

The Key Idea

Avoid Candidate Generation!

Apriori suffers from generating huge numbers of candidates. FP-Growth adopts a **Divide-and-Conquer** strategy.

- Compress DB into an FP-tree.
- Mine the tree recursively.



FP-Growth Steps

First Scan

Build a frequency list of items (like Apriori's L_1).

Second Scan

Build the FP-tree (Frequent Pattern tree). This is a compressed representation of the database.

Mine Recursively

Build conditional pattern bases and construct conditional FP-trees for each item.

Building the FP-Tree

Compression Strategy

The FP-tree retains the itemset association information but compresses shared paths.

Sorting Logic

Items in a transaction are processed in **frequency descending order** (F-List).

Example Logic:

Transaction: {f, a, c, d, g, i, m, p}

↓ Sort by Freq

Sorted: **{f:4, c:4, a:3, m:3, p:3}**

(Assume d, g, i are infrequent)

Example: Database & Frequencies

Transaction Database

T1	{f, a, c, d, g, i, m, p}
T2	{a, b, c, c, f, l, m, o}
T3	{b, f, h, j, o}
T4	{b, c, k, s, p}
T5	{a, f, c, e, l, p, m, n}

Frequency Analysis

min_sup = 3

Sorted Frequent Items (L):

{f:4} {c:4} {a:3} {b:3} {m:3} {p:3}

The FP-Tree Structure

Items are inserted following the sorted order. Shared prefixes (like f, c, a) are merged, counts incremented.

Mining the FP-Tree

Recursive Approach

We don't generate candidates. We explore the tree starting from the bottom (least frequent items).

1. Construct Conditional Pattern Base
2. Build Conditional FP-Tree
3. Mine Recursively

Advantages Over Apriori

2 Scans Only

Regardless of pattern length, we only scan the DB twice.

No Candidates

Eliminates the expensive candidate generation step.

Speed

Much faster, especially for dense datasets with long patterns.

Mining Using Vertical Data Format

Table 4.3 The vertical data format of the transaction data set *D* of Table 4.1.

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}

Table 4.4 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}

Table 4.5 3-Itemsets in vertical data format.

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

Data Formats: A Comparison

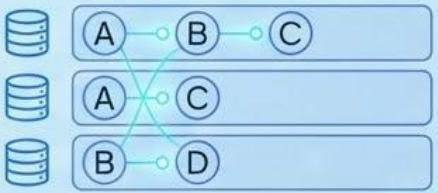
↔Horizontal (Traditional)

Structure: Transaction ID → Itemset

T1: A, B, C

T2: A, C

T3: B, D



Used by Apriori & FP-Growth

TRANSFORM

↕Vertical (Eclat)

Structure: Item → TID-List

A: {T1, T2}

B: {T1, T3}

C: {T1, T2}

D: {T3}

Inverts the data structure

The Eclat Algorithm

Eclat = Equivalence Class Transformation

Vertical Format

Uses **TID-lists** (Transaction ID lists) for every item instead of scanning rows.

Intersection

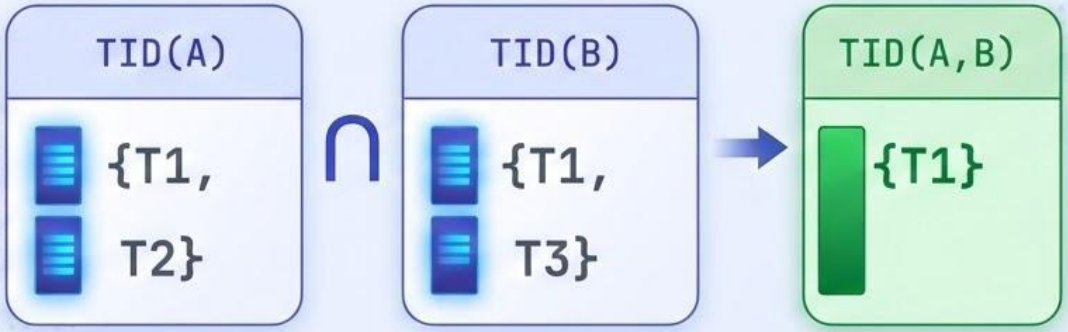
Computes support by **intersecting** TID-lists. No counting scan required.

Depth-First

Uses **Depth-First Search (DFS)** traversal strategy to find frequent itemsets.

Example: Computing Support

How do we find the support of itemset **{A, B}**?



Support({A, B}) = Count of Result = **1**

Advantages of Eclat

Fast Support Counting

Uses set intersection logic, which is computationally faster than pattern matching.

Sparse Data

Excellent for sparse datasets where itemsets are small.

Memory Efficient

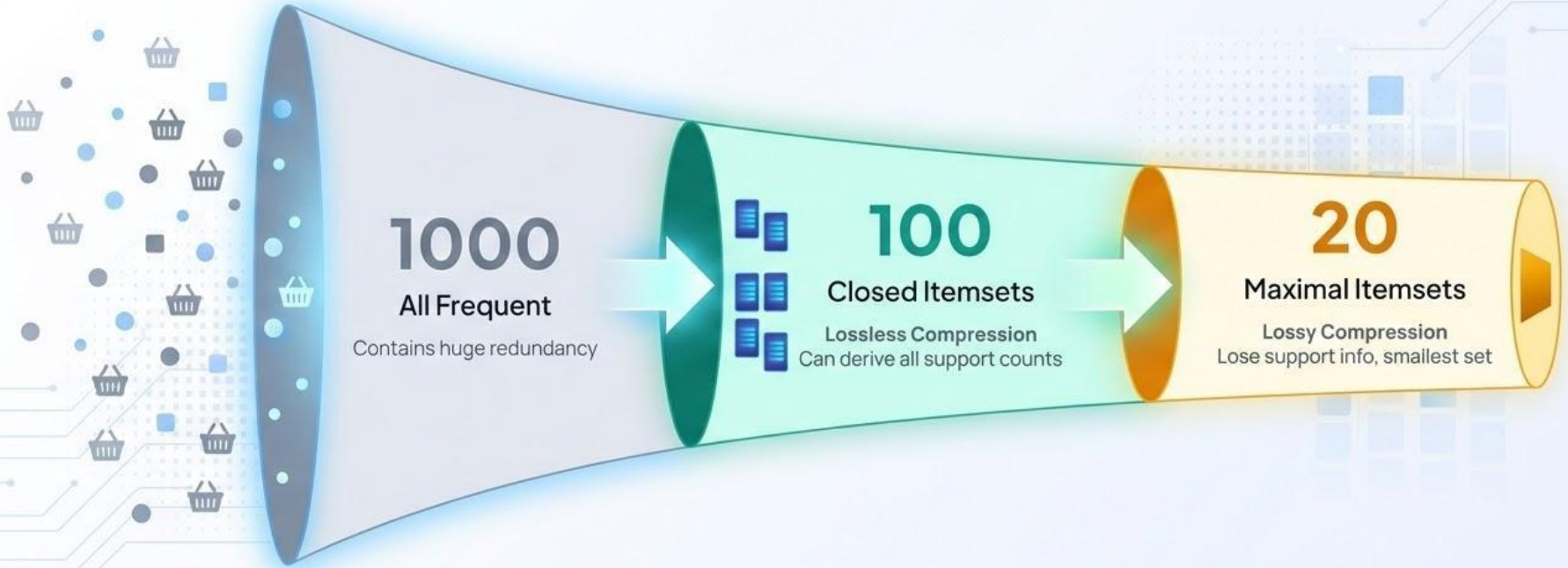
Depth-first search strategy reduces memory consumption during traversal.

No Candidate Generation

(Similar benefit to FP-Growth)

Why Mine Closed/Maximal?

Drastic reduction in the number of patterns generated.

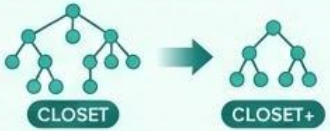


Closed Itemset Algorithms

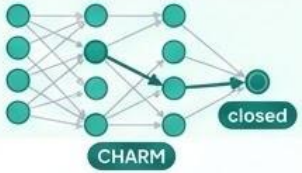
Methodologies

Apriori-based:
Modify standard Apriori by adding a closure checking step.

FP-Growth based:
Algorithms like **CLOSET** and **CLOSET+** optimize the FP-tree structure for closure.



Vertical Format: CHARM
explores the itemset-tid lattice to skip non-closed sets.



Lossless Compression

Maximal Itemset Algorithms

Key Approaches

Maximum Compactness

MaxMiner: Uses *lookahead pruning*. Before checking a branch, it estimates if the branch can possibly yield a longer frequent pattern.

GenMax: Uses *backtracking search* to quickly identify maximal patterns without generating all subsets.

⚖ The Practical Rule

EXACT ANALYSIS

Mine **Closed Patterns** when you need complete support information for exact analysis.

HIGH-LEVEL OVERVIEW

Mine **Maximal Patterns** when you need a quick, high-level overview of the largest frequent combinations.

Strong Rules Are Not Necessarily Interesting

Example 4.7. A misleading "strong" association rule. Suppose we are interested in analyzing transactions with respect to the purchase of computer games and videos. Let *game* refer to the transactions containing computer games, and *video* refer to those containing videos. Of the 10,000 transactions analyzed, the data show that 6000 of the customer transactions included computer games, whereas 7500 included videos, and 4000 included both computer games and videos. Suppose that a data mining program for discovering association rules is run on the data, using a minimum support of, say, 30% and a minimum confidence of 60%. The following association rule is discovered:

$$buys(X, \text{“}computer\ games\text{”}) \Rightarrow buys(X, \text{“}videos\text{”})$$
$$[support = 40\%, confidence = 66\%]. \tag{4.6}$$

Rule (4.6) is a strong association rule and would therefore be reported, since its support value of $\frac{4000}{10{,}000} = 40\%$ and confidence value of $\frac{4000}{6000} = 66\%$ satisfy the minimum support and minimum confidence thresholds, respectively. However, Rule (4.6) is misleading because the probability of purchasing videos is 75%, which is even larger than 66%. In fact, computer games and videos are negatively associated because the purchase of one of these items actually decreases the likelihood of purchasing the other. Without fully understanding this phenomenon, we could easily make unwise business decisions based on Rule (4.6). □

The Data Setup

We analyze **1000 transactions** with the following counts:

800
COFFEE

600
TEA

400
COFFEE & TEA (BOTH)

Mining Thresholds

min_sup = 0.3 (30%)

30% 30%

min_conf = 0.6 (60%)

60% 60%

Evaluating Potential Rules

Wait... Is it really interesting?

Rule 1: Coffee → Tea

Support: 400/1000 = 0.4 (≥ 0.3) ✓

Confidence: 400/800 = 0.5 (< 0.6) ✗

Result: **Not Strong.**

Rule 2: Tea → Coffee

Support: 0.4 (≥ 0.3) ✓

Confidence: 400/600 = 0.667 (≥ 0.6) ✓

Result: **STRONG RULE!**

Misleading Association!

The rule **Tea → Coffee** is "Strong" (66.7% confidence).
However, let's look at the overall probability of buying **Coffee**. Buying **Tea** actually reduces the probability of buying Coffee (from 80% down to 66.7%).

They are **negatively correlated**. Overall **P(Coffee) > P(Coffee | Tea)**

High Confidence is Not Enough

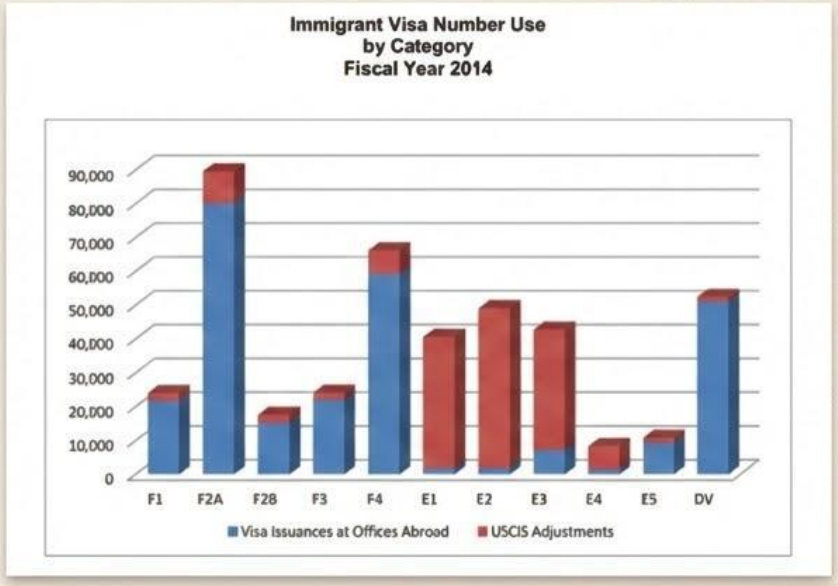
Problem:

If the consequent (Y) is already very frequent, a high confidence rule X → Y might still be uninteresting or misleading.

Solution:

We need to measure **Lift** or **Correlation** to check if X actually *increases* the likelihood of Y.

Data Example



The Solution: Measure Correlation

Why Association isn't enough

High confidence rules can be misleading if the consequent is already very frequent.

The Fix: We must calculate Lift (Interest Factor)

Lift measures how much the occurrence of X raises the probability of occurring Y.

Visualizing Correlation Types

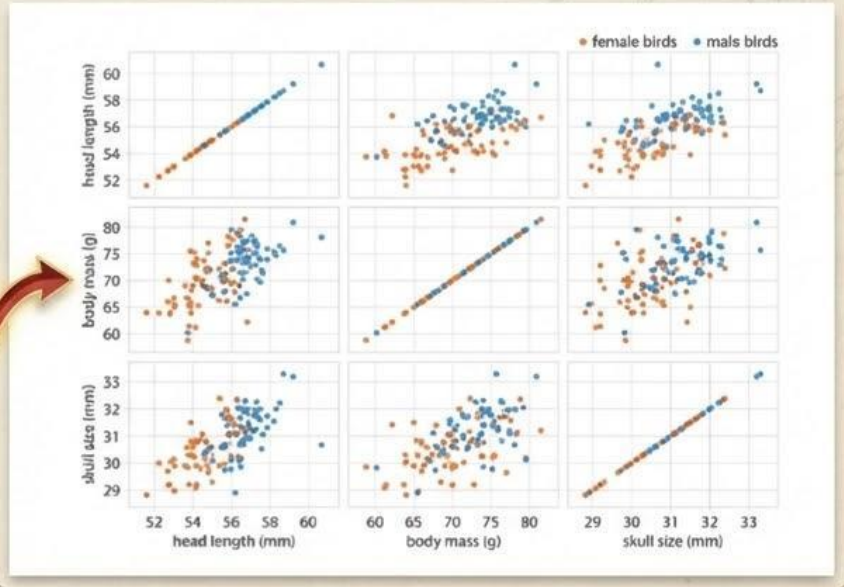


Table 4.6 2×2 **contingency table summarizing the transactions with respect to game and video purchases.**

	$game$	$\overline{game}$	Σ_{row}
$video$	4000	3500	7500
$\overline{video}$	2000	500	2500
Σ_{col}	6000	4000	10,000

Table 4.7 Table 4.6 contingency table, now with the expected values.

	$game$	$\overline{game}$	Σ_{row}
$video$	4000 (4500)	3500 (3000)	7500
$\overline{video}$	2000 (1500)	500 (1000)	2500
Σ_{col}	6000	4000	10,000

Example 4.9. Correlation analysis using χ^2. To compute the correlation using χ^2 analysis for nominal data, we need the observed value and expected value (displayed in parenthesis) for each slot of the contingency table, as shown in Table 4.7. From the table, we can compute the χ^2 value as follows:

$$\chi^2 = \Sigma \frac{(observed - expected)^2}{expected} = \frac{(4000 - 4500)^2}{4500} + \frac{(3500 - 3000)^2}{3000}$$
$$+ \frac{(2000 - 1500)^2}{1500} + \frac{(500 - 1000)^2}{1000} = 555.6.$$

Because the χ^2 value is greater than 1, and the observed value of the slot (*game*, *video*) $= 4000$, which is less than the expected value of 4500, *buying game* and *buying video* are *negatively correlated*. This is consistent with the conclusion derived from the analysis of the *lift* measure in Example 4.8. □

Lift: Formula & Meaning

$$\text{Lift}\,(A \rightarrow B) = \frac{P(A \cup B)}{P(A) * P(B))}$$

= 1

Independent

X and Y appear together by chance. No relationship.

> 1

Positive

Positively correlated. X makes Y more likely.

< 1

Negative

Negatively correlated. X makes Y less likely.

Revisiting Coffee & Tea

Let’s apply Lift to our paradoxical rule:

Rule: Tea ⇢ Coffee

- P(Tea) = 0.6
- P(Coffee) = 0.8
- P(Tea ∪ Coffee) = 0.4

Calculation

Lift = 0.4 / (0.6 × 0.8)

Lift = 0.4 / 0.48

Lift = 0.833

0.833 < 1

Negative Correlation!

(Despite having high confidence)

Other Correlation Measures

Chi-Square (χ^2)

A standard statistical test to check if the observed frequency significantly differs from the expected frequency (independence).

All-Confidence

$$\min\{P(X|Y), P(Y|X)\}$$

Ensures the rule holds strong in *both* directions.

Cosine

$$\frac{P(X \cup Y)}{\sqrt{P(X) * P(Y)}}$$

Geometric mean of lift-like logic. Good for vector analysis.

Popular Measures Overview

Measure	Formula	Range	Null Value	Sym?
Support	$P(X \cup Y)$	[0, 1]	0	Yes
Confidence	$P(Y \mid X) = \frac{P(X \cup Y)}{P(X)}$	[0, 1]	$P(Y)$	No
Lift	$\frac{P(Y \mid X)}{P(Y)} = \frac{P(X \cup Y)}{P(X)P(Y)}$	[0, ∞)	1	Yes
Conviction	$\frac{P(X)P(\neg Y)}{P(X \cup \neg Y)}$	[0, ∞)	1	No
Convince	$P(Y \mid X) - P(Y)$	[-1, 1]	0	Yes
Cosine	$\frac{P(X \cup Y)}{\sqrt{P(X)P(Y)}}$	[0, 1]	0	Yes
Jaccard	$\frac{P(X \cup Y)}{P(X)+P(Y)-P(X \cup Y)}$	[0, 1]	0	Yes

Table 4.8 2 × 2 contingency table for two items.

	$milk$	$\overline{milk}$	Σ_{row}
$coffee$	mc	$\overline{m}c$	c
$\overline{coffee}$	$m\overline{c}$	$\overline{mc}$	$\overline{c}$
Σ_{col}	m	$\overline{m}$	Σ

Table 4.9 Comparison of six pattern evaluation measures using contingency tables for a variety of data sets.

Data Set	mc	$\overline{m}c$	$m\overline{c}$	$\overline{m}\,\overline{c}$	χ^2	*lift*	*all_conf.*	*max_conf.*	*Kulc.*	*cosine*
D_1	10,000	1000	1000	100,000	90,557	9.26	0.91	0.91	0.91	0.91
D_2	10,000	1000	1000	100	0	1	0.91	0.91	0.91	0.91
D_3	100	1000	1000	100,000	670	8.44	0.09	0.09	0.09	0.09
D_4	1000	1000	1000	100,000	24,740	25.75	0.5	0.5	0.5	0.5
D_5	1000	100	10,000	100,000	8173	9.18	0.09	0.91	0.5	0.29
D_6	1000	10	100,000	100,000	965	1.97	0.01	0.99	0.5	0.10

Key Properties

Null-Invariant

Not affected by the number of null transactions (¬X ∩ ¬Y).

YES: Cosine, Jaccard, All-Conf

NO: Lift, Correlation

Anti-Monotonic

If a set fails the test, all supersets fail. Crucial for pruning.

YES: Support

NO: Confidence

Symmetry

Is the relationship X → Y the same as Y → X?

YES: Lift, Cosine, Jaccard

NO: Confidence, Conviction

Guidelines for Choosing

Actionable Rules (Marketing)
Use **Confidence + Lift** to find strong, directional implications.

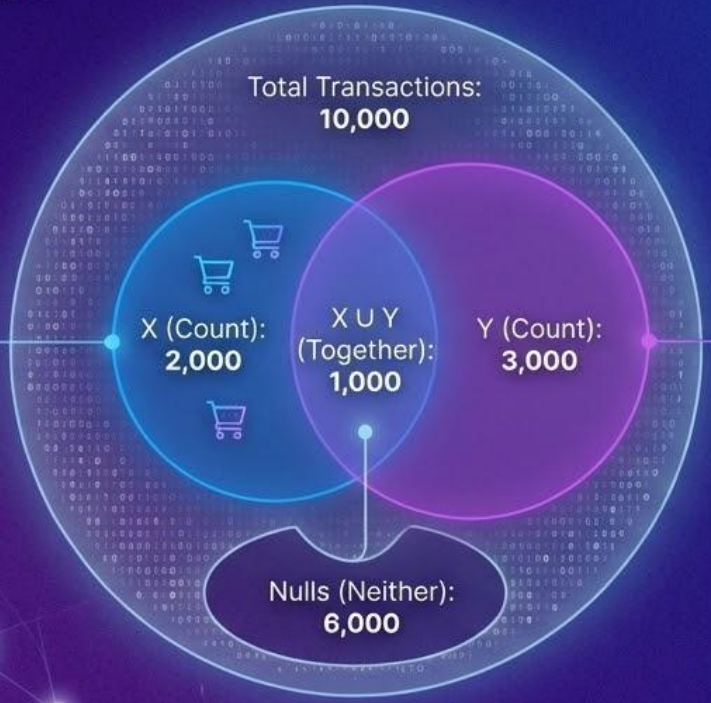
Correlated Pairs
Use **Cosine** or **Jaccard** when null transactions don't matter.

Statistical Validity
Use **Chi-Square (χ^2)** to test independence formally.

Null-Invariant Needs
Use **All-confidence** when dataset is sparse and nulls are noise.

Example Comparison

The Data



Calculations

- Support **0.1**
- Confidence 0.1 / 0.2 = **0.5**
- Lift 0.1 / (0.2 × 0.3) = **1.67**
- Cosine 0.1 / √(0.06) = **0.408**
- Jaccard 0.1 / (0.2+0.3-0.1) = **0.25**

Algorithm Selection Guide

Apriori

- Small number of items
- Sparse datasets
- Simple implementation needed

FP-Growth

- Dense datasets
- Many frequent patterns
- Performance critical (2 scans)

Eclat

- Sparse datasets
- Vertical format available
- Memory efficient (DFS)

Practical Considerations

min_sup setting:

Too high → miss patterns.
Too low → combinatorial explosion.

min_conf setting:

Balance between interestingness and data coverage.

Validation:

Always check **Lift/Correlation** to avoid misleading rules.

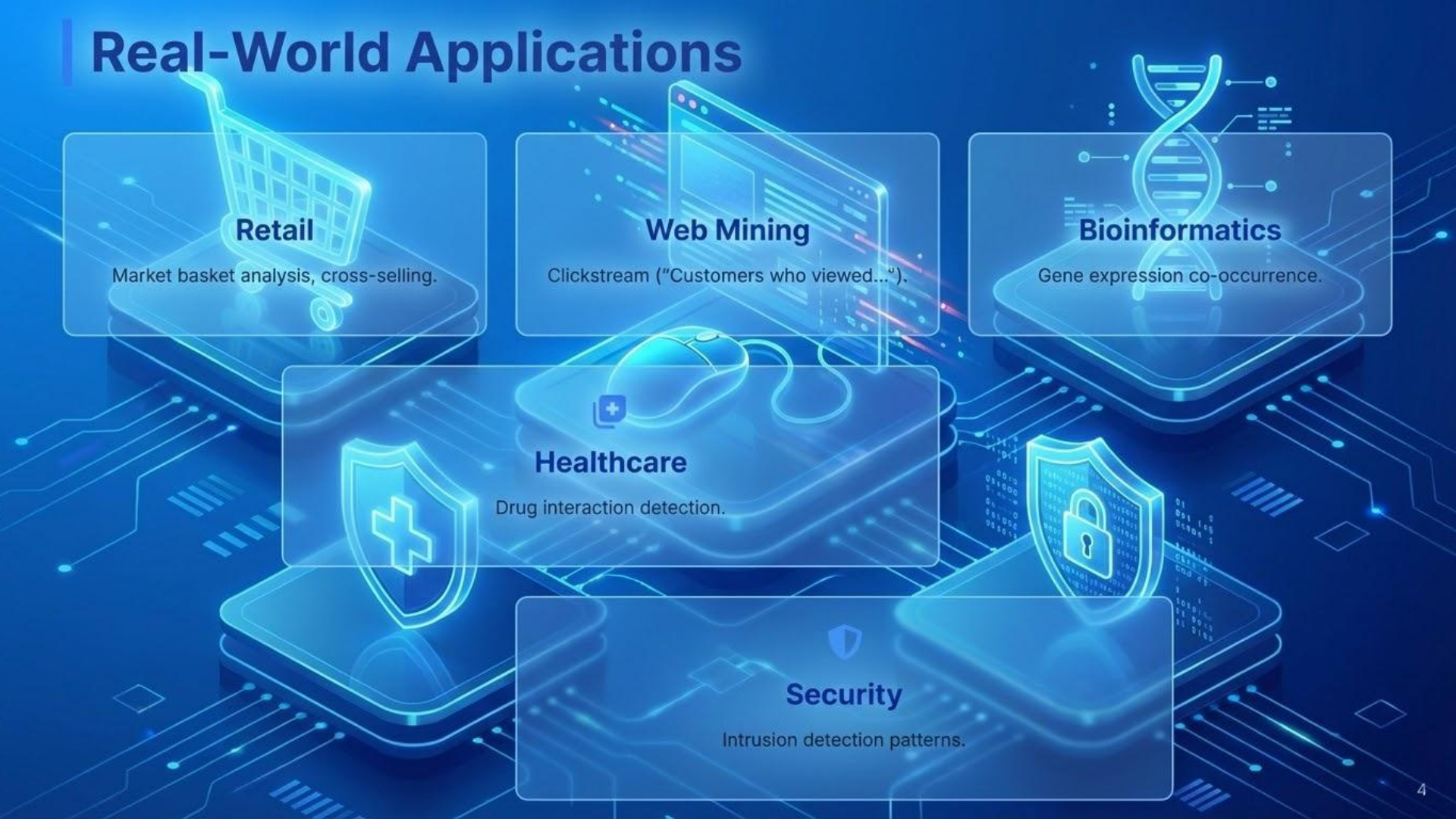
Efficiency:

Consider mining **Closed** or **Maximal** patterns.

Find the Balance

Support vs. Significance

Real-World Applications



Retail

Market basket analysis, cross-selling.

Web Mining

Clickstream (“Customers who viewed…“).

Bioinformatics

Gene expression co-occurrence.

Healthcare

Drug interaction detection.

Security

Intrusion detection patterns.

Common Pitfalls

Ignoring Lift

Finding "strong" rules (high confidence) that are actually negatively correlated.

Multiple Comparison Problem

With thousands of items, some random correlations will appear statistically significant by chance.

Temporal Blindness

Ignoring time factors. (e.g., "Beer and diapers" might only correlate on Friday evenings).

⚠ Association ≠ Causation