

The background of the slide is a complex, abstract composition. It features a dark, reddish-brown base with a network of thin, light-colored lines forming a mesh or web-like structure. Scattered throughout are numerous small, colored dots in shades of green, blue, and orange. In the upper left, there's a horizontal band with a grid of small, light-colored plus signs. In the lower left, there's a rectangular inset showing a cluster of orange and red dots. The overall aesthetic is technical and data-driven.

Session 2: The Apriori Algorithm

Apriori: A Candidate Generation & Test Approach

- ❑ Outline of Apriori (level-wise, candidate generation and test)
 - ❑ Initially, scan DB once to get frequent 1-itemset
 - ❑ Repeat
 - ❑ Generate length-(k+1) candidate itemsets from length-k frequent itemsets
 - ❑ Test the candidates against DB to find frequent (k+1)-itemsets
 - ❑ Set $k := k + 1$
 - ❑ Until no frequent or candidate set can be generated
 - ❑ Return all the frequent itemsets derived

The Apriori Algorithm (Pseudo-Code)

C_k : Candidate itemset of size k

F_k : Frequent itemset of size k

$K := 1$;

$F_k := \{\text{frequent items}\}$; // frequent 1-itemset

While ($F_k \neq \emptyset$) **do** { // when F_k is non-empty

$C_{k+1} := \text{candidates generated from } F_k$; // candidate generation

 Derive F_{k+1} by counting candidates in C_{k+1} with respect to TDB at minsup;

$k := k + 1$

}

return $\cup_k F_k$ // return F_k generated at each level

The Apriori Algorithm—An Example

Database TDB

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

minsup = 2

C_1

1st scan

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

F_1

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

C_2

F_2

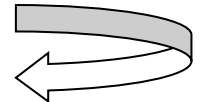
Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

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

2nd scan

C_2

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



C_3

Itemset	sup
{B, C, E}	2

3rd scan

F_3

Itemset	sup
{B, C, E}	2

Apriori: Implementation Tricks

□ How to generate candidates?

□ Step 1: self-joining F_k

□ Step 2: pruning

□ Example of candidate-generation

□ $F_3 = \{abc, abd, acd, ace, bcd\}$

□ Self-joining: $F_3 * F_3$

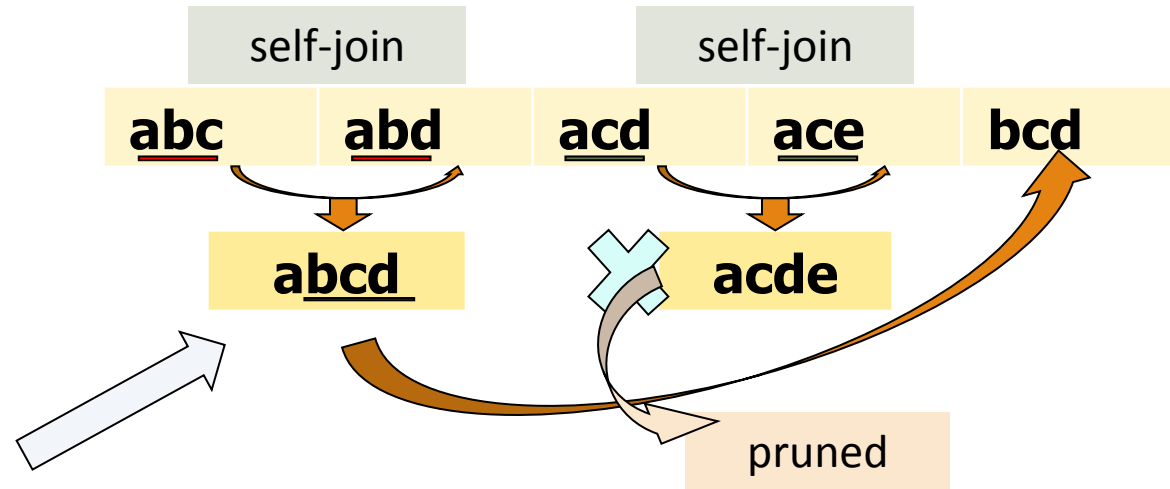
□ $abcd$ from abc and abd

□ $acde$ from acd and ace

□ Pruning:

□ $acde$ is removed because ade is not in F_3

□ $C_4 = \{abcd\}$



Candidate Generation: An SQL Implementation

- Suppose the items in F_{k-1} are listed in an order

- Step 1: self-joining F_{k-1}
insert into C_k

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

from F_{k-1} as p, F_{k-1} as q

where $p.item_1 = q.item_1, \dots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$

- Step 2: pruning

for all *itemsets* c in C_k do

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

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

