

# XML & Intelligente Systeme

## - XQuery Teil 2 - Datamining auf XML Dokumenten

Seminarvortrag von  
Dominik Westhues

*dwesthue@techfak.uni-bielefeld.de*

21.11.2005

# XML & Intelligente Systeme

## Überblick

- XQuery & Datamining
- Verknüpfungsregeln
- Apriori Algorithmus
- Verknüpfungsregel Extraktion

# XML & Intelligente Systeme

## Warum XQuery für Data Mining?

- Normalerweise: Vor- /Nachverarbeitung nötig
- Hier: Wie kann man *ohne* Vor- /Nachverarbeitung Daten extrahieren?
  - Also: Data Mining *direkt* auf XML Daten, keine separate Programmiersprache und andere Datenformate nötig
  - Datenaufbereitung = 90 % der gesamten Verarbeitungszeit
- Verknüpfungsregeln (*association rules*) extrahieren, also das Finden von Verknüpfungsarten und Assoziationen von XML Elementen

# XML & Intelligente Systeme

## Was sind Verknüpfungsregeln?

- Berühmte, überraschende Verknüpfungsregel: „**Wenn** Bier gekauft wird, **dann** werden oft auch Windeln gekauft.“
- Basisansatz:
  - Gesamtmenge von Items (Brot, Butter, Milch, etc)
  - Transaktionen sind Teilmengen daraus, quasi (z.B.) „Einkäufe“
  - Verknüpfungsregeln sind Implikationen
  - Wenn Menschen Brot und Butter kaufen, kaufen sie zu 66% auch Milch.
  - 80% der Transaktionen, die Brot und Butter enthalten, enthalten Milch.

# XML & Intelligente Systeme

## Verknüpfungsregeln bei Einkäufen

T Nr.	Item
1	{Brot, Butter, Milch}
2	{Brot, Butter, Milch, Eiskrem}
3	{Eiskrem, Cola}
4	{Batterien, Brot, Butter, Milch}
5	{Brot, Butter, Milch}
6	{Batterien, Eiskrem, Brot, Butter}

- $I = \{\text{Brot, Butter, Milch, Eiskrem, Cola, Batterien}\}$ ,  $T1 = \{\text{Brot, Butter, Milch}\} \subset I$
- VR 'Brot, Butter  $\Rightarrow$  Milch'
- Support : Brot,Butter und Milch-Einkäufe/Transaktionen vs. alle Einkäufe
- $support('Brot,Butter \Rightarrow \text{Milch}') = 4/6 = 0.66 = 66\%$
- Confidence: Bei wievielen Brot und Butter Einkäufen wurde auch Milch eingekauft?
- $confidence('Brot,Butter \Rightarrow \text{Milch}') = 4/5 = 0.8 = 80\%$
- Also: VR ('Brot, Butter  $\Rightarrow$  Milch' | support = 0.66 | confidence = 0.8)

# XML & Intelligente Systeme

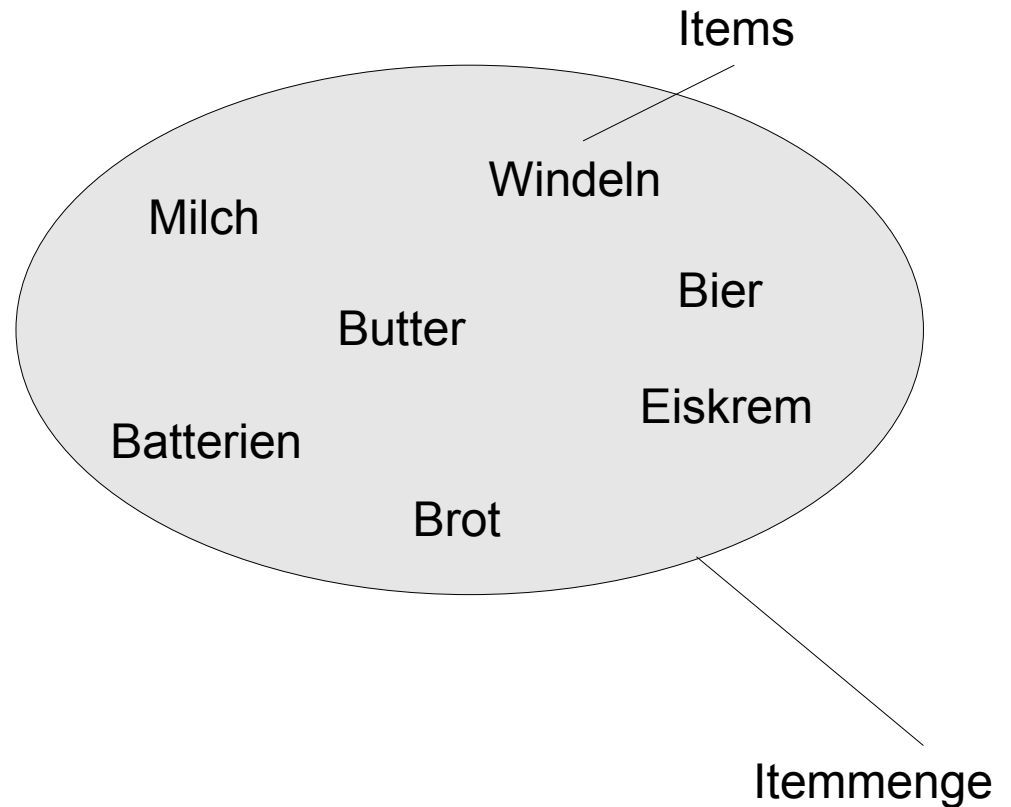
## Large Itemsets

- Man kann enorm viele Regeln finden, vor allem auch unnütze / redundante.
- Ansatz: Minimum für confidence (*minconf*) & support (*minsupp*) festlegen.  
→ nur noch VRs mit *minconf* und *minsupp* wichtig

Eine Itemmenge / Transaktionsmenge heisst „**Large Itemset**“ ( $I_G$ ):  
alle Regeln haben **hinreichenden Support** ( $\geq \text{minsupp}$ ).

# XML & Intelligente Systeme

## Der Apriori Algorithmus

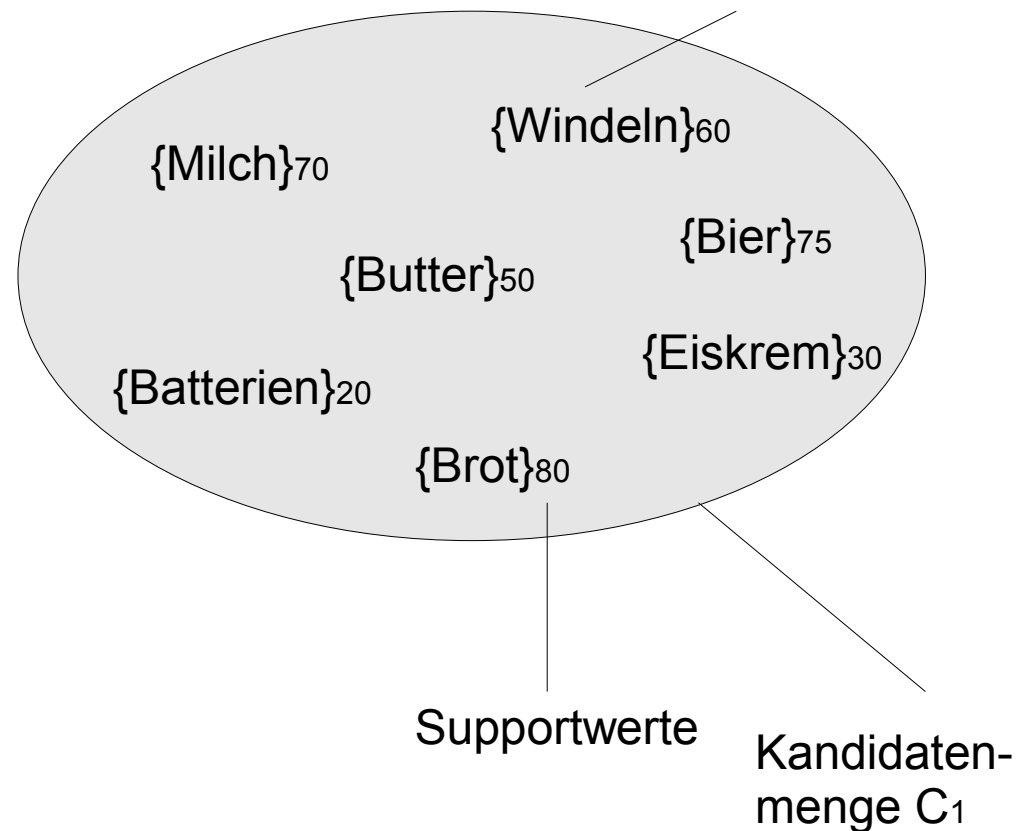


# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$

1-elem.  
Kandidaten



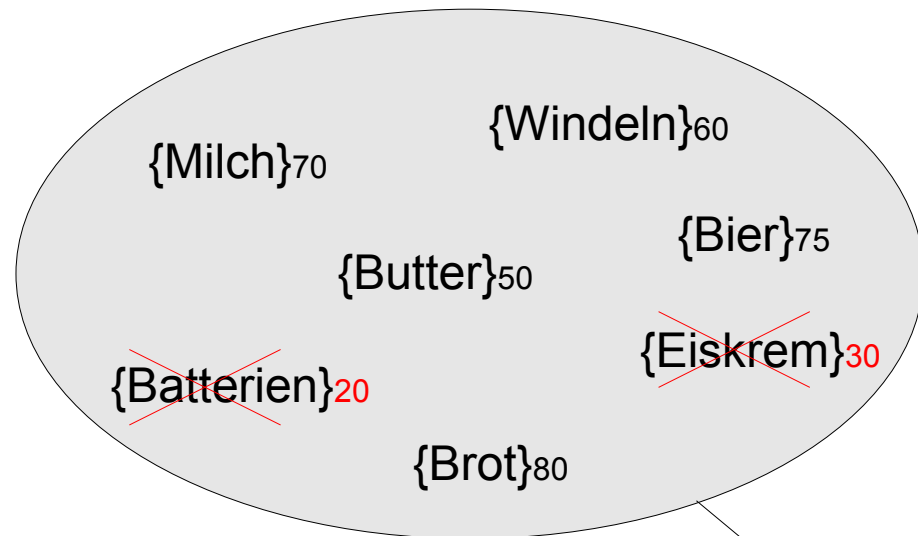
- Anfangs-Large-Itemset  $L_0 = \emptyset$
- $\text{Minsupp} := 40\% = 0.4$
- Generiere aus der Itemmenge die Kandidatenmenge  $C_1$
- Fasse alle Items in Itemmenge als 1-elem. Kandidaten mit jeweiligen Supportwerten auf

# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }

- **Prune Schritt**
- Entferne alle Kandidaten aus  $C_1$  mit **zu kleinen** Support Werten (unter 40)
- Übrig bleibt  $L_1$ , die Menge der Large Itemsets in  $C_1$



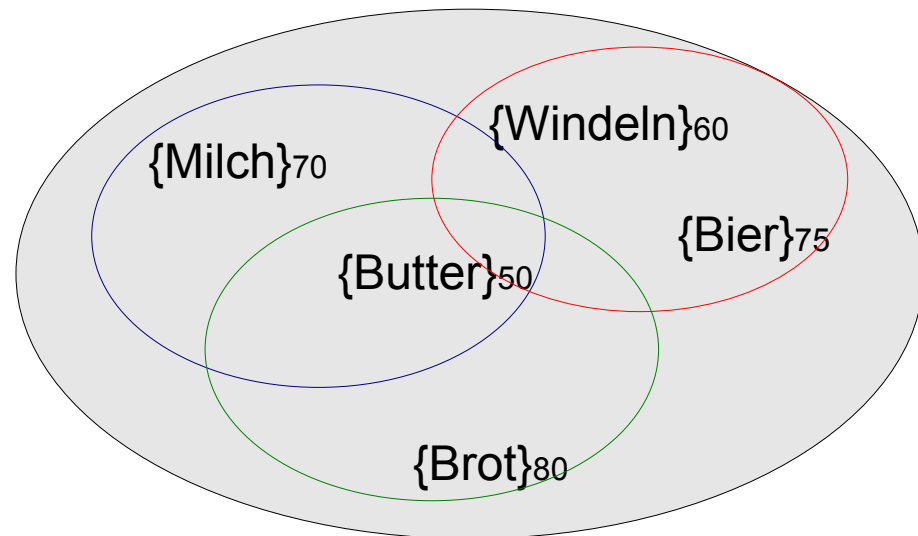
$L_1$  = Menge der Large Itemsets in  $C_1$

# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }

- Nächster Schritt: **join**
- Falls  $L_1$  nicht leer ist:
- Generiere **2**-elementige Kandidaten in die Kandidatenmenge  $C_2$  durch Kombinieren (**joinen**) der Large Itemsets in  $L_1$ , mit jeweils neuen Supportwerten

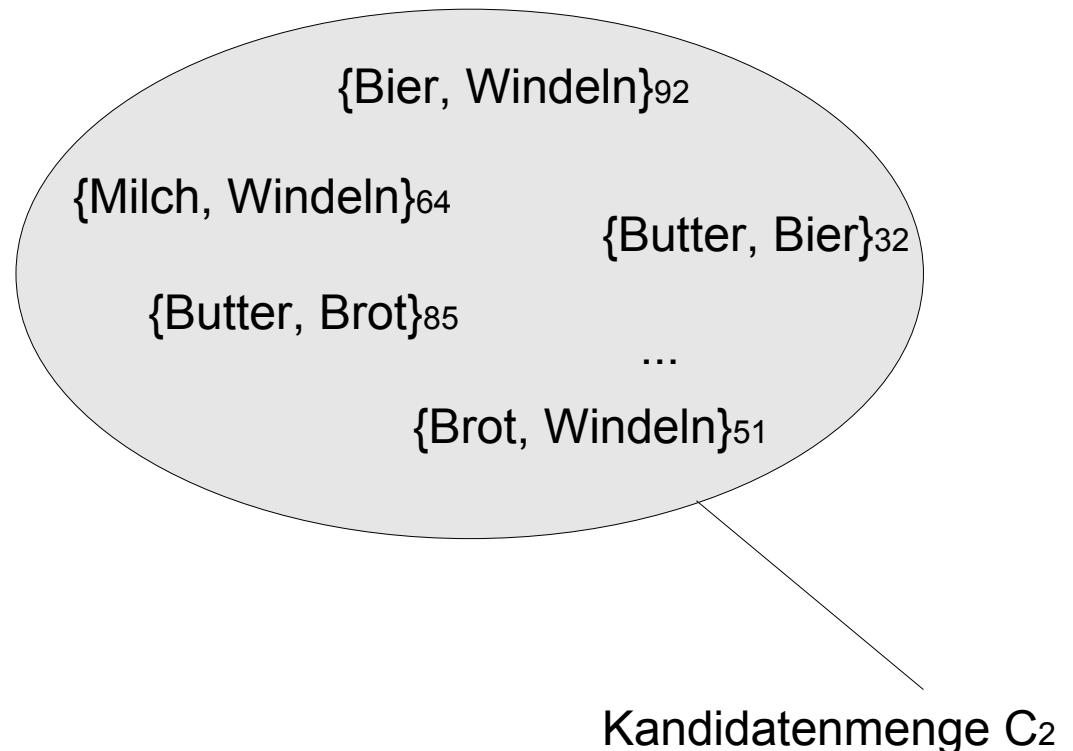


# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }

- Im nächsten Schritt:
- Falls  $L_1$  nicht leer ist:
- Generiere 2-elementige Kandidaten in die Kandidatenmenge  $C_2$  durch Kombinieren (**joinen**) der Large Itemsets in  $L_1$ , mit jeweils neuen Supportwerten

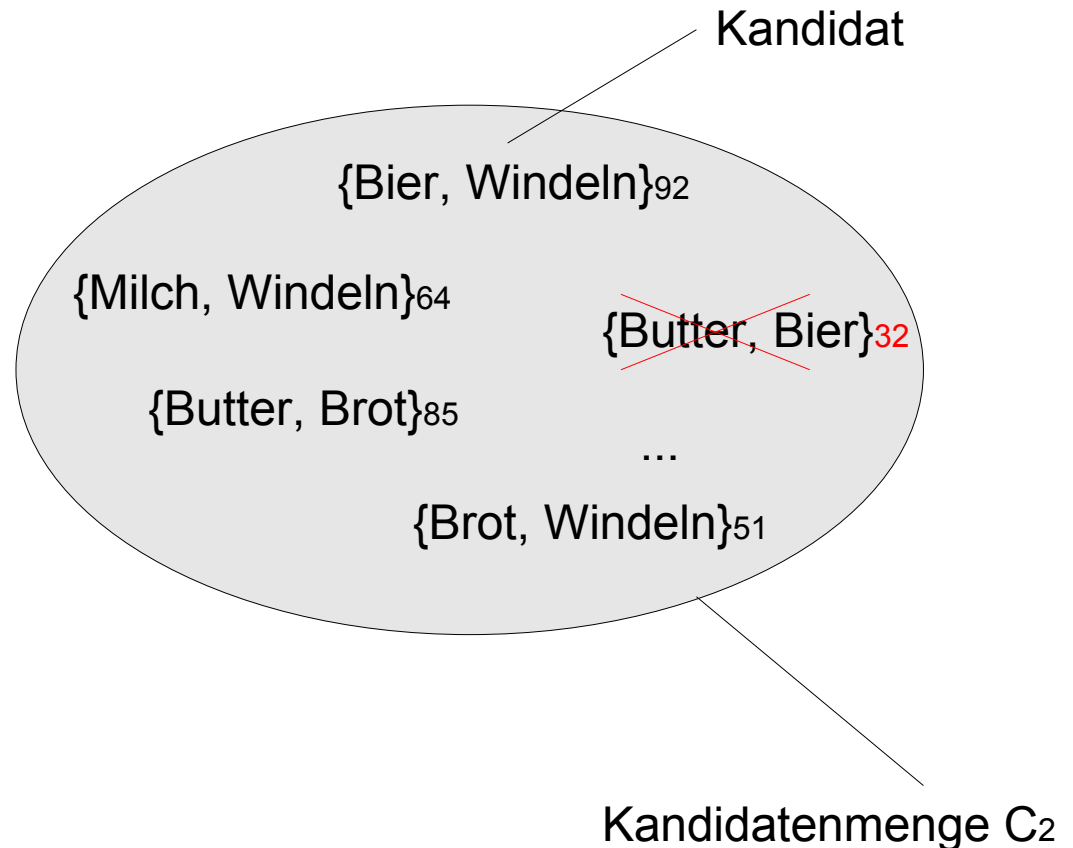


# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }
2	{ {Bier, Windeln} , ... }

- Entferne (**prune**) alle Kandidaten aus  $C_2$  mit **zu kleinen** Support Werten (unter 40)
- Übrig bleibt  $L_2$ , die Menge der Large Itemsets in  $C_2$

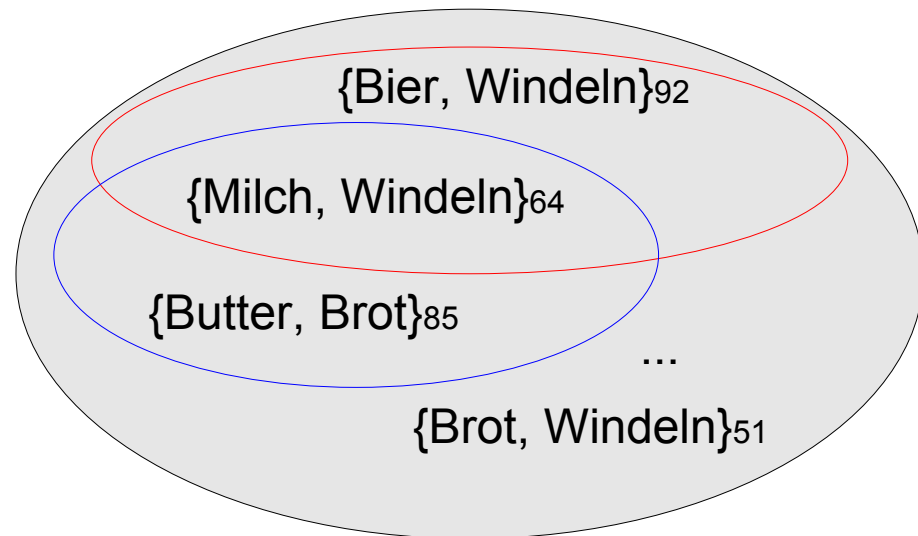


# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }
2	{ {Bier, Windeln} , ... }

- Im nächsten Schritt:
- Falls  $L_2$  nicht leer ist:
- Generiere 3-elementige Kandidaten in die Kandidatenmenge  $C_3$  durch Kombinieren (**joinen**) der Large Itemsets in  $L_2$

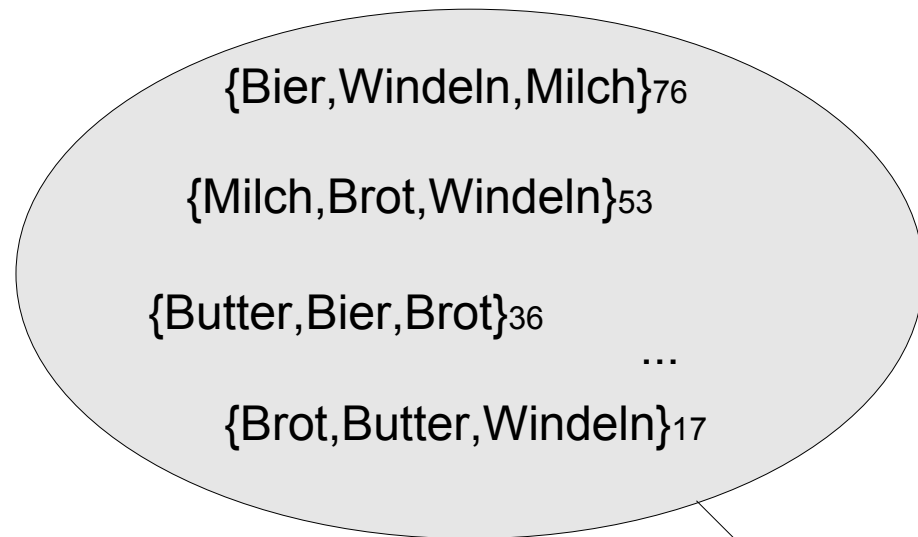


# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }
2	{ {Bier, Windeln} , ... }

- Im nächsten Schritt:
- Falls  $L_2$  nicht leer ist:
- Generiere 3-elementige Kandidaten in die Kandidatenmenge  $C_3$  durch Kombinieren (**joinen**) der Large Itemsets in  $L_2$



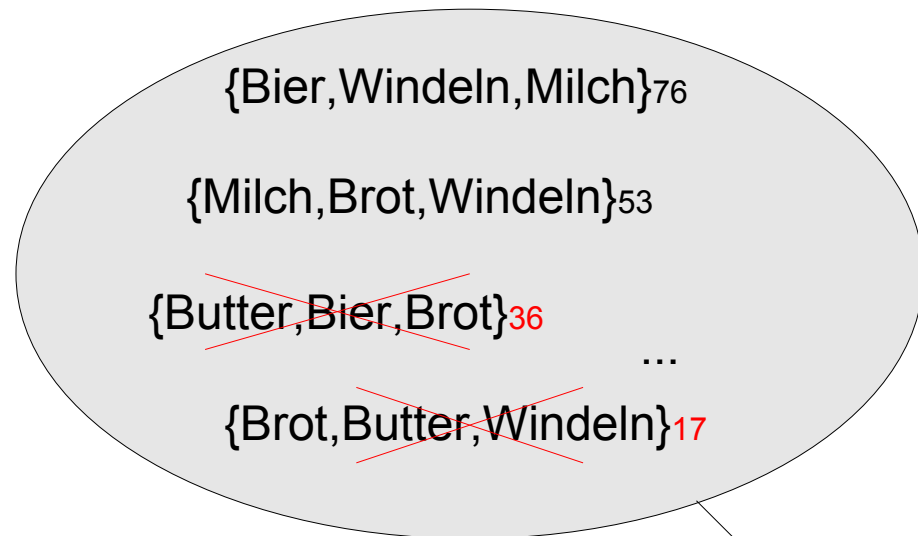
Kandidatenmenge  $C_3$

# XML & Intelligente Systeme

## Der Apriori Algorithmus

K	Large Itemsets $L_k$
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }
2	{ {Bier, Windeln} , ... }
3	{ {Bier, Windeln, Milch}, ... }

- Entferne (**prune**) alle Kandidaten aus  $C_2$  mit **zu kleinen** Support Werten (unter 40)
- Übrig bleibt  $L_3$ , die Menge der Large Itemsets in  $C_3$
- usw...



Kandidatenmenge  $C_3$

# XML & Intelligente Systeme

## Der Apriori Algorithmus

<b>K</b>	<b>Large Itemsets <math>L_k</math></b>
0	$\emptyset$
1	{ {Milch}, {Windeln}, ... }
2	{ {Bier, Windeln}, ... }
3	{ {Bier, Windeln, Milch}, ... }

- Dies geht soweit bis die Menge  $L_k$  im k.ten Schritt (spätestens bei  $k = n$ ) leer ist, (keine Kandidaten mehr vorhanden waren, alle Supportwerte waren unter 40%, oder keine neuen Kandidaten erstellbar)
- RETURN: Die Vereinigung aller  $L_k$

# XML & Intelligente Systeme

## Der Apriori Algorithmus

- Benutzt Bottom-Up Breadth-First Ansatz für mehr Effizienz:
  - sonst: kombinatorisch viele Kandidaten erzeugt/“gejoint“ und auf hinreichenden Support testen (SEHR viele Vergleiche, viele Zugriffe auf die DB teuer)
  - So aber: prunen aller nicht hinreichenden Kandidaten, aus diesen werden keine neuen Kandidaten erzeugt.

# XML & Intelligente Systeme

## APRIORI Anwendung in XQuery

```
let $src := document(„transactions.xml“) //source document with items
let $minsupp := 0.4
```

# XML & Intelligente Systeme

## APRIORI Anwendung in XQuery

```
let $src := document(„transactions.xml“)  
let $minsupp := 0.4  
let $total := count($src) * 1.00    //count all items/transactions
```

# XML & Intelligente Systeme

## APRIORI Anwendung in XQuery

```
let $src := document(„transactions.xml“)  
let $minsupp := 0.4  
let $total := count($src) * 1.00  
let $C := distinct-values($src/*) //generate C1 = 1-elm. Candidateset
```

# XML & Intelligente Systeme

## APRIORI Anwendung in XQuery

```
let $src := document („transactions.xml“)
let $minsupp := 0.4
let $total := count($src) * 1.00
let $C := distinct-values($src/*)

//typischer XQuery FLWR Ausdruck
let $L := (for $itemset in $C/*           //generate L1 = LargeList
           let $items :=(for $item in $src/*
                          where $itemset = $item
                          return $item)
           let $sup := (count($items) * 1.00) div $total
           where $sup >= $minsupp       //remove items <= 40%
           return <largeItemset>
                <items> {$itemset} </items>
                <support> {$sup} </support>
           </largeItemset>)
```

# XML & Intelligente Systeme

## APRIORI Anwendung in XQuery

```
let $src := document („transactions.xml“)
let $minsupp := 0.4
let $total := count($src) * 1.00
let $C := distinct-values($src/*)

//typischer XQuery FLWR Ausdruck
let $L := (for $itemset in $C/*
           let $items :=(for $item in $src/*
                        where $itemset = $item
                        return $item)
           let $sup := (count($items) * 1.00) div $total
           where $sup >= $minsupp
           return <largeItemset>
                   <items> {$itemset} </items>
                   <support> {$sup} </support>
           </largeItemset>)
let $UL := $L //Füge 1-elem Large Itemset der Gesamtmenge zu
return <largeItemsets>
       {apriori($L,$UL,$minsupp,$total,$src)}
       </largeItemsets> //berechne rekursiv 2,3,... elem. Lk's
```

# XML & Intelligente Systeme

## Die Apriori Funktion in XQuery

```
define function apriori( element $L, element $UL,           //Input
                        element $minsupp, element $total
                        element $src ) returns element    //Ergebnis
{
  //redundante, mehrfach auftretende Kombinationen löschen
  let $C := removeDuplicate( candidateGen($L) )

  //aus ($C = Kandidatenmenge) ein Large Itemset machen2
  let $L := getLargeItemsets( $C,$minsupp,$total,$src )

  // $UL = Menge aller Large Itemsets vereinigen mit k-elem. $L
  let $UL := $L union $UL

  //Abbruchkriterium: $L ist leer, sonst rekursiver Aufruf
  return if ( empty($L) ) then $UL
           else apriori( $L,$UL,$minsupp, $total,$src )
}
```

<sup>2</sup> für mehr Informationen zu *getLargeItemsets()* siehe [4]

# XML & Intelligente Systeme

## transactions.xml

```
<transactions>
  <transaction id="1">
    <item>Brot</item>
    <item>Windeln</item>
    <item>Bier</item>
  </transaction>
  <transaction id="2">
    <item>Bier</item>
    <item>Brot</item>
    <item>Butter</item>
    <item>Windeln</item>
  </transaction>
  .
  .
  ....
</transactions>
```

Source Dokument

Apriori

## largeItemSets.xml

```
<largeItemSets>
  <largeItemSet>
    <items>Windeln</items>
    <support>0.7</support>
  </largeItemSet>
  <largeItemSet>
    <items>Bier</items>
    <support>0.75</support>
  </largeItemSet>
  <largeItemSet>
    <items>Bier,Windeln</items>
    <support>0.9</support>
  </largeItemSet>
  ...
</largeItemSets>
```

1  
elem  
lg's

2 elem  
lg's

Ziel Dokument

# XML & Intelligente Systeme

## Rules Extraction in XQuery

```
let $minconf := 1.00
let $src := document(„/largeItemSets.xml/“)
for $itemset1 in $src
  let $items1 := $itemset1/items/*
  for $itemset2 in $src
    let $items2 := $itemset2/items/*
    where count($items1) > count($items2) and
      count(commonIts($items1, $items2) =
        count($items2) and $itemset1/support div
          $itemset2/support ≥ $minconf
  return <rule
    support = „{$itemset1/support}“
    confidence = „{$itemset1/support * 1.00} div
      ($itemset2/support * 1.00)}“>
    <antecedent> {$items2} </antecedent>
    <consequent> {removeItems($items1, $items2)}
    </consequent>
  </rule>
```

# XML & Intelligente Systeme

## Ergebnis der Rule Extraction

```
<rules>
```

```
<rule support=„0.9“ confidence=„1.0“>  
  <antecedent>  
    <item>Bier</item>  
  </antecedent>  
  <consequent>  
    <item>Windeln</item>  
  </consequent>  
</rule>
```

Wer Windeln kauft, kauft auch Bier.

```
<rule support=„0.66“ confidence=„1.0“>  
  <antecedent>  
    <item>Brot</item>  
    <item>Butter</item>  
  </antecedent>  
  <consequent>  
    <item>Milch</item>  
  </consequent>  
</rule>
```

Wer Brot und Butter kauft,  
kauft auch Milch.

```
</rules>
```

# XML & Intelligente Systeme

## Literatur

- [1] *Mining Association Rules from XML Data using Xquery*,  
J. W.W. Wan, Gillian Dobbie
- [2] World Wide Web Consortium. *XQuery 1.0: An XML Query Language*  
(W3C Working: An XML Query Language (W3C Working Draft).  
*<http://www.w3.org/TR/2002/WDxquery-20020816>*, Aug. 2002.
- [3] XQuery, Wikipedia – The Free Encyclopedia, [www.en.wikipedia.org/XQuery](http://www.en.wikipedia.org/XQuery)
- [4] *Extracting association rules from XML documents using Xquery*, J.W.W.Wan and  
G.Dobbie, In „*Proceedings of Fifth International Workshop on Web  
Information and Data Management*“, New Orleans, LA

# XML & Intelligente Systeme

## Fragen und Diskussion

- Wie steht es mit der Performance?

# XML & Intelligente Systeme

## Der Apriori Algorithmus

Pseudo Code:

**Input:** Datenbank DB, minsupp

**Output:** alle Grossen Itemlisten als Menge von  $I_G$ 's

- 1) Anfangs: keine  $I_G$ 's,  $L_0 = \emptyset$ ,  $k = 0$ .
- 2)  $C_1 :=$  alle unterschiedlichen Items in DB
- 3)  $L_1 :=$  alle Grossen Listen in  $C_1$
- 4) **WHILE** ( $L_{k+1} \neq \emptyset$ )
- 5)      $C_{k+1} =$  Kandidaten-Generierung( $L_k$ ) [\[Join\]](#)
- 6)      $L_{k+1} =$  alle Grossen Listen in  $C_{k+1}$  [\[Prune\]](#)
- 7)      $k++$
- 8) **return** (Vereinigung aller  $L_k$ )