

# Mining XML Functional Dependencies through Formal Concept Analysis

Viorica Varga

May 1, 2010

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

# Outline

Definitions for XML Functional Dependencies

Introduction to FCA

FCA tool to detect XML FDs

Finding XML keys

Detecting XML data redundancy

Conclusions and Future Work

## XML Design

- ▶ XML data design: choose an appropriate XML schema, which usually come in the form of DTD (Document Type Definition) or XML Schema.
- ▶ Functional dependencies (FDs) are a key factor in XML design.
- ▶ The objective of normalization is to eliminate redundancies from an XML document, eliminate or reduce potential update anomalies.
- ▶ Arenas, M., Libkin, L.: A normal form for XML documents. TODS 29(1), 195-232 (2004)
- ▶ Yu, C., Jagadish, H. V.: XML schema refinement through redundancy detection and normalization. VLDB J. 17(2): 203-223 (2008)



# Schema definition

## Definition

(Schema) A schema is defined as a set  $S = (E, T, r)$ , where:

- ▶  $E$  is a finite set of element labels;
- ▶  $T$  is a finite set of element types, and each  $e \in E$  is associated with a  $\tau \in T$ , written as  $(e : \tau)$ ,  $\tau$  has the next form:  
 $\tau ::= \mathbf{str} \mid \mathbf{int} \mid \mathbf{float} \mid \mathbf{SetOf} \tau \mid \mathbf{Rcd} [e_1 : \tau_1, \dots, e_n : \tau_n];$
- ▶  $r \in E$  is the label of the root element, whose associated element type can not be **SetOf**  $\tau$ .
  
- ▶ Types **str**, **int** and **float** are the system defined *simple types* and **Rcd** indicate *complex scheme elements*.
- ▶ Keyword **SetOf** is used to indicate *set schema elements*
- ▶ Attributes and elements are treated in the same way, with a reserved "@" symbol before attributes.

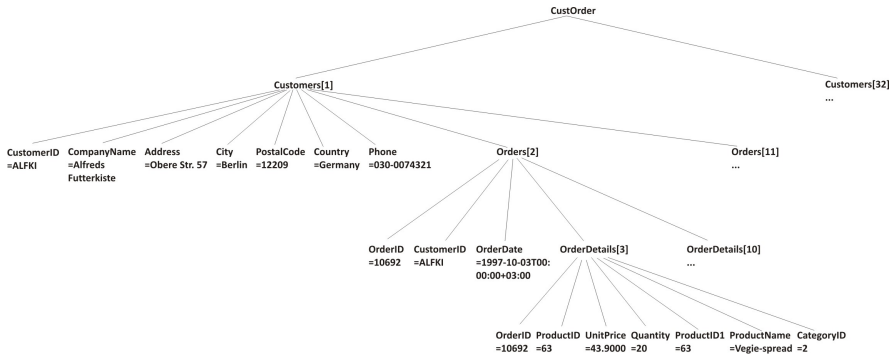


Figure: CustOrder XML tree

## Example scheme

CustOrder:Rcd

Customers:SetOf Rcd

CustomerID: str

CompanyName: str

Address: str

City: str

PostalCode:str

Country: str

Phone: str

Orders: SetOf Rcd

OrderID: int

CustomerID: str

OrderDate: str

OrderDetails: SetOf Rcd

OrderID: int

ProductID: int

UnitPrice: float

Quantity: float

- ▶ A schema element  $e_k$  can be identified through a path expression,  $path(e_k) = /e_1/e_2/.../e_k$ , where  $e_1 = r$ , and  $e_i$  is associated with type  $\tau_i ::= \mathbf{Rcd} [\dots, e_{i+1} : \tau_{i+1}, \dots]$  for all  $i \in [1, k - 1]$ .
- ▶ A path is *repeatable*, if  $e_k$  is a set element. We adopt XPath steps "." (self) and ".." (parent)

**Definition** (Data tree) An XML database is defined to be a rooted labeled tree  $T = \langle N, \mathcal{P}, \mathcal{V}, n_r \rangle$ , where:

- ▶  $N$  is a set of labeled data nodes, each  $n \in N$  has a label  $e$  and a node key that uniquely identifies it in  $T$ ;
- ▶  $n_r \in N$  is the root node;
- ▶  $\mathcal{P}$  is a set of parent-child edges, there is exactly one  $p = (n', n)$  in  $\mathcal{P}$  for each  $n \in N$  (except  $n_r$ ), where  $n' \in N, n \neq n', n'$  is called the parent node,  $n$  is called the child node;
- ▶  $\mathcal{V}$  is a set of value assignments, there is exactly one  $v = (n, s)$  in  $\mathcal{V}$  for each leaf node  $n \in N$ , where  $s$  is a value of simple type.

## Descendant, repeatable element definition

- ▶ We assign a node key, referred to as @key, to each data node in the data tree in a pre-order traversal.
- ▶ A data element  $n_k$  is a descendant of another data element  $n_1$  if there exists a series of data elements  $n_i$ , such that  $(n_i, n_{i+1}) \in \mathcal{P}$  for all  $i \in [1, k - 1]$ .
- ▶ Data element  $n_k$  can be addressed using a path expression,  $path(n_k) = /e_1/ \dots /e_k$ , where  $e_i$  is the label of  $n_i$  for each  $i \in [1, k]$ ,  $n_1 = n_r$ , and  $(n_i, n_{i+1}) \in \mathcal{P}$  for all  $i \in [1, k - 1]$ .
- ▶ A data element  $n_k$  is called **repeatable** if  $e_k$  corresponds to a *set element* in the schema.
- ▶ Element  $n_k$  is called a *direct descendant* of element  $n_a$ , if  $n_k$  is a descendant of  $n_a$ ,  $path(n_k) = \dots /e_a/e_1/ \dots /e_{k-1}/e_k$ , and  $e_i$  is not a set element for any  $i \in [1, k - 1]$ .

**Definition**(Element-value equality) Two data elements  $n_1$  of  $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1} \rangle$  and  $n_2$  of  $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2} \rangle$  are *element-value equal* (written as  $n_1 =_{ev} n_2$ ) if and only if:

- ▶  $n_1$  and  $n_2$  both exist and have the same label;
- ▶ There exists a set  $M$ , such that for every pair  $(n'_1, n'_2) \in M$ ,  $n'_1 =_{ev} n'_2$ , where  $n'_1, n'_2$  are children elements of  $n_1, n_2$ , respectively. Every child element of  $n_1$  or  $n_2$  appears in exactly one pair in  $M$ .
- ▶  $(n_1, s) \in \mathcal{V}_1$  if and only if  $(n_2, s) \in \mathcal{V}_2$ , where  $s$  is a simple value.

**Definition**(Path-value equality) Two data element paths  $p_1$  on  $T_1 = \langle N_1, \mathcal{P}_1, \mathcal{V}_1, n_{r1} \rangle$  and  $p_2$  on  $T_2 = \langle N_2, \mathcal{P}_2, \mathcal{V}_2, n_{r2} \rangle$  are *path-value equal* (written as  $T_1.p_1 =_{pv} T_2.p_2$ ) if and only if there is a set  $M'$  of matching pairs where

- ▶ For each pair  $m' = (n_1, n_2)$  in  $M'$ ,  $n_1 \in N_1$ ,  $n_2 \in N_2$ ,  $path(n_1) = p_1$ ,  $path(n_2) = p_2$ , and  $n_1 =_{ev} n_2$ ;
- ▶ All data elements with path  $p_1$  in  $T_1$  and path  $p_2$  in  $T_2$  participate in  $M'$ , and each such data element participates in only one such pair.

## Generalized tree tuple

**Definition** A generalized tree tuple of data tree  $T = \langle N, \mathcal{P}, \mathcal{V}, n_r \rangle$ , with regard to a particular data element  $n_p$  (called pivot node), is a tree  $t_{n_p}^T = \langle N^t, \mathcal{P}^t, \mathcal{V}^t, n_r \rangle$ , where:

- ▶  $N^t \subseteq N$  is the set of nodes,  $n_p \in N^t$  ;
- ▶  $\mathcal{P}^t \subseteq \mathcal{P}$  is the set of parent-child edges;
- ▶  $\mathcal{V}^t \subseteq \mathcal{V}$  is the set of value assignments;
- ▶  $n_r$  is the same root node in both  $t_{n_p}^T$  and  $T$ ;
- ▶  $n \in N^t$  if and only if:
  - ▶  $n$  is a descendant or ancestor of  $n_p$  in  $T$ , or
  - ▶  $n$  is a non-repeatable direct descendant of an ancestor of  $n_p$  in  $T$  ;
- ▶  $(n_1, n_2) \in \mathcal{P}^t$  if and only if  $n_1 \in N^t$ ,  $n_2 \in N^t$ ,  $(n_1, n_2) \in \mathcal{P}$ ;
- ▶  $(n, s) \in \mathcal{V}^t$  if and only if  $n \in N^t$ ,  $(n, s) \in \mathcal{V}$ .

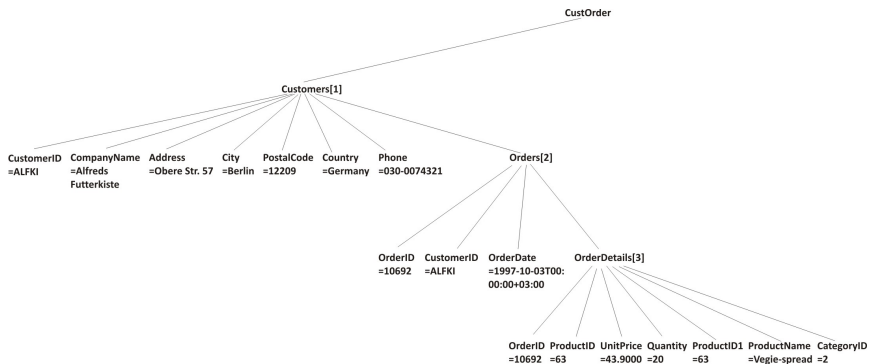


Figure: Example tree tuple



- ▶ A generalized tree tuple is a data tree projected from the original data tree.
- ▶ It has an extra parameter called a pivot node. In contrast with tree tuple defined in Arenas and Libkin's article, which separate sibling nodes with the same path at all hierarchy levels, the generalized tree tuple separate sibling nodes with the same path above the pivot node
- ▶ Based on the pivot node, generalized tree tuples can be categorized into tuple classes:

**Definition**(Tuple class) A tuple class  $C_p^T$  of the data tree  $T$  is the set of all generalized tree tuples  $t_n^T$ , where  $path(n) = p$ . Path  $p$  is called the *pivot path*.

**Definition**(XML FD) An XML FD is a triple  $\langle C_p, LHS, RHS \rangle$ , written as  $LHS \rightarrow RHS$  w.r.t.  $C_p$ , where  $C_p$  denotes a tuple class,  $LHS$  is a set of paths  $(P_{li}, i = [1, n])$  relative to  $p$ , and  $RHS$  is a single path  $(P_r)$  relative to  $p$ .

An XML FD holds on a data tree  $T$  (or  $T$  satisfies an XML FD) if and only if for any two generalized tree tuples  $t_1, t_2 \in C_p$

- $\exists i \in [1, n], t_1.P_{li} = \perp$  or  $t_2.P_{li} = \perp$ , or
- If  $\forall i \in [1, n], t_1.P_{li} =_{pv} t_2.P_{li}$ , then  $t_1.P_r \neq \perp, t_2.P_r \neq \perp, t_1.P_r =_{pv} t_2.P_r$ .

A null value,  $\perp$ , results from a path that matches no node in the tuple, and  $=_{pv}$  is the path-value equality defined previous.

## Example

(XML FD) In our running example whenever two products agree on ProductID values, they have the same ProductName . This can be formulated as follows:

$./ProductID \rightarrow ./ProductName$  w.r.t  $C_{OrderDetails}$

Another example is:

$./ProductID \rightarrow ./CategoryID$  w.r.t  $C_{OrderDetails}$

## XML key

**Definition** (XML key) An XML Key of a data tree  $T$  is a pair  $\langle C_p, LHS \rangle$ , where  $T$  satisfies the XML FD  $\langle C_p, LHS, ./@key \rangle$ .

### Example

We have the XML FD:  $\langle C_{Orders}, ./OrderID, ./@key \rangle$ , which implies that  $\langle C_{Orders}, ./OrderID \rangle$  is an XML key.

Tuple classes with repeatable pivot paths are called *essential tuple classes*.

**Definition**(Interesting XML FD) An XML FD  $\langle C_p, LHS, RHS \rangle$  is *interesting* if it satisfies the following conditions:

- ▶  $RHS \notin LHS$ ;
- ▶  $C_p$  is an essential tuple class;
- ▶  $RHS$  matches to descendent(s) of the pivot node.

An interesting XML FD is a non-trivial XML FD with an essential tuple class

- ▶ **Definition**(XML data redundancy) A data tree  $T$  contains a redundancy if and only if  $T$  satisfies an interesting XML FD  $\langle C_p, LHS, RHS \rangle$ , but does not satisfy the XML Key  $\langle C_p, LHS \rangle$ .

# Introduction to FCA

- ▶ From a philosophical point of view a concept is a unit of thoughts consisting of two parts:
  - ▶ the extension, which are objects;
  - ▶ the intension consisting of all attributes valid for the objects of the context;
- ▶ Formal Concept Analysis (FCA) introduced by Wille gives a mathematical formalization of the concept notion.
- ▶ A detailed mathematic foundation of FCA can be found in:
  - ▶ Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer, Berlin-Heidelberg-New York. (1999)
- ▶ Formal Concept Analysis is applied in many different realms like psychology, sociology, computer science, biology, medicine and linguistics.
- ▶ FCA is a useful tool to explore the conceptual knowledge contained in a database by analyzing the formal conceptual structure of the data.

- ▶ FCA studies how objects can be hierarchically grouped together according to their common attributes. In FCA the data is represented by a cross table, called formal context.
- ▶ A *formal context* is a triple  $(G, M, I)$ .
- ▶  $G$  is a finite set of objects
- ▶  $M$  is finite set of attributes
- ▶ The relation  $I \subseteq G \times M$  is a binary relation between objects and attributes.
- ▶ Each couple  $(g, m) \in I$  denotes the fact that the object  $g \in G$  is related to the item  $m \in M$ .

- ▶ For a set  $A \subseteq G$  of objects we define

$$A' := \{m \in M \mid gIm \text{ for all } g \in A\}$$

the set of all attributes common to the objects in  $A$ .

- ▶ Dually, for a set  $B \subseteq M$  of attributes we define

$$B' := \{g \in G \mid gIm \text{ for all } m \in B\}$$

the set of all objects which have all attributes in  $B$ .

- ▶ A *formal concept* of the context  $\mathbb{K} := (G, M, I)$  is a pair  $(A, B)$  where  $A \subseteq G$ ,  $B \subseteq M$ ,  $A' = B$ , and  $B' = A$ .
- ▶ We call  $A$  the *extent* and  $B$  the *intent* of the concept  $(A, B)$ .
- ▶ The set of all concepts of the context  $(G, M, I)$  is denoted by  $\mathfrak{B}(G, M, I)$ .

## Example formal context

- ▶ The following cross table describes for some hotels the attributes they have.
- ▶ In this case the objects are: *Oasis*, *Royal*, *Amelia*, *California*, *Grand*, *Samira*;
- ▶ and the attributes are: *Internet*, *Sauna*, *Jacuzzi*, *ATM*, *Babysitting*.
- ▶  $(\{California, Grand\})' := \{Sauna, Jacuzzi\}$ .

	Internet	Sauna	Jacuzzi	ATM	Babysitting
Oasis	X	X	X	X	
Royal	X	X	X		
Amelia	X				X
California		X	X		
Grand		X	X		
Samira				X	X

Table: Formal context of the Hotel facilities example



## FCA tool to detect XML FDs

- ▶ we elaborate an FCA based tool that identify functional dependencies in XML documents.
- ▶ to achieve this, as a first step, we have to construct the Formal Context of functional dependencies for XML data.
- ▶ we have to identify the objects and attributes of this context in case of XML data.
- ▶ tuple-based XML FD notion proposed in the above section suggests a natural technique for XFD discovery
- ▶ XML data can be converted into a fully unnested relation, a single relational table, and apply existing FD discovery algorithms directly.
- ▶ given an XML document, which contains at the beginning the schema of the data, we create generalized tree tuples from it.

## Construct Formal Context of XML FDs


- ▶ each tree tuple in a tuple class has the same structure, so it has the same number of elements.
- ▶ we use the flat representation which converts the generalized tree tuples into a flat table
- ▶ each row in the table corresponds to a tree tuple in the XML tree
- ▶ in the flat table we insert non-leaf and leaf level elements (or attributes) from the tree
- ▶ for non-leaf level nodes the associated keys are used as values
- ▶ we include non-leaf level nodes with associated key values, to detect XML keys

# Flat table for tuple class $C_{Orders}$

## Example

Let us construct the flat table for tuple class  $C_{Orders}$ . There are two non-leaf nodes:

- ▶ Orders, appears as Orders@key
- ▶ OrderDetails, appears as OrderDetails@key.



Orders/Orders@key	Orders/OrderID	Orders/Customer	Orders/OrderDate	OrderDetails/OrderDetails@key	OrderDetails/OrderID	OrderDetails/ProductID	OrderDetails/UnitPrice	OrderDetails/Quantity	OrderDetails/ProductName	OrderDetails/CategoryID
2	10643	ALFKI	1997-08-25T00...	3	10643	28	45.6000	15	Rossle Sauerkraut	7
2	10643	ALFKI	1997-08-25T00...	4	10643	39	18.0000	21	Chartreuse verte	1
2	10643	ALFKI	1997-08-25T00...	5	10643	46	12.0000	2	Spagessild	8
6	10692	ALFKI	1997-10-03T00...	7	10692	63	43.9000	20	Veggie-spread	2
8	10702	ALFKI	1997-10-13T00...	9	10702	3	10.0000	6	Aniseed Syrup	2
8	10702	ALFKI	1997-10-13T00...	10	10702	76	18.0000	15	Lakkalikoon	1
11	10835	ALFKI	1998-01-15T00...	12	10835	59	55.0000	15	Raclette Courdevault	4

## Formal Context for class $C_{Orders}$

- ▶ Context's Attributes: *PathEnd/ElementName*
  - ▶ for non-leaf level nodes: the name of the attribute is constructed as:  $\langle \text{ElementName} \rangle + "@key"$  and its value will be the associated key value
  - ▶ for non-leaf level nodes: the element names of the leaves.
- ▶ Context's Objects: the objects are considered to be the tree tuple pairs, actually the tuple pairs of the flat table. The key values associated to non-leaf elements and leaf element's values are used in these tuple pairs.
- ▶ Context's Properties: the mapping between objects and attributes is defined by a binary relation, this incidence relation of the context shows which attributes of this tuple pairs have the same value.

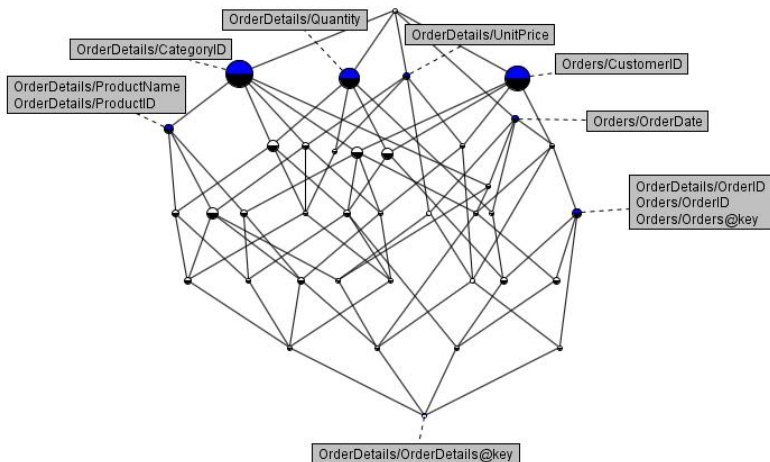
## Beginning of the Formal Context of functional dependencies for tuple class $C_{Orders}$

	Orders/Orders@key	Orders/OrderID	Orders/CustomerID	Orders/OrderDate	OrderDetails/OrderDetails@key	OrderDetails/OrderID
2,10643,ALFKI,1997-08-25T00:....	⊗	⊗	⊗	⊗		⊗
2,10643,ALFKI,1997-08-25T00:....	⊗	⊗	⊗	⊗		
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			
2,10643,ALFKI,1997-08-25T00:....			⊗			

- ▶ the analyzed XML document may have a large number of tree tuples.
- ▶ we filter the tuple pairs and we leave out those pairs in which there are no common attributes, by an operation called "clarifying the context", which does not alter the conceptual hierarchy.

# Concept Lattice of functional dependencies' Formal Context for tuple class $C_{Orders}$

- ▶ we run the Concept Explorer (ConExp) engine to generate the concepts and create the concept lattice.



# Processing the Output of FCA

- ▶ a concept lattice consists of the set of concepts of a formal context and the subconcept-superconcept relation between the concepts;
- ▶ every circle represents a formal concept;
- ▶ each concept is a tuple of a set of objects and a set of common attributes, but only the attributes are listed;
- ▶ an edge connects two concepts if one implies the other directly;
- ▶ each link connecting two concepts represents the transitive subconcept-superconcept relation between them;
- ▶ the top concept has all formal objects in its extension;
- ▶ the bottom concept has all formal attributes in its intension.

# The relationship between FDs in databases and implications in FCA

a FD  $X \rightarrow Y$  holds in a relation  $r$  over  $R$  iff the implication  $X \rightarrow Y$  holds in the context  $(G, R, I)$  where  $G = \{(t_1, t_2) \mid t_1, t_2 \in r, t_1 \neq t_2\}$  and  $\forall A \in R, (t_1, t_2)IA \Leftrightarrow t_1[A] = t_2[A]$ .

- ▶ objects of the context are couples of tuples and each object intent is the agree set of this couple
- ▶ the implications in this lattice corresponds to functional dependencies in XML.

## Example

$\langle C_{Orders}, ./OrderID, ./CustomerID \rangle$

$\langle C_{Orders}, ./Orders@key, ./CustomerID \rangle$

$\langle C_{Orders}, ./OrderDetail/OrderID, ./CustomerID \rangle$



## Reading the Concept Lattice

- ▶ in the lattice we list only the attributes, these are relevant for our analysis;
- ▶ let there be a concept, labeled by  $A, B$  and a second concept labeled by  $C$ , where  $A, B$  and  $C$  are FCA attributes;
- ▶ let concept labeled by  $A, B$  be the subconcept of concept labeled by  $C$ ;
- ▶ tuple pairs of concept labeled by  $A, B$  have the same values for attributes  $A, B$ , but for attribute  $C$  too.
- ▶ tuple pairs of concept labeled by  $C$  do not have the same values for attribute  $A$ , nor for  $B$ , but have the same value for attribute  $C$ .
- ▶ tuple pairs of every subconcept of concept labeled by  $A, B$  have the same values for attributes  $A, B$ .
- ▶ the labeling of the lattice is simplified by putting each attribute only once, at the highest level.

# Reading the Concept Lattice

- ▶ we analyze attributes  $A$  and  $B$ :
  - ▶ if we have only  $A \rightarrow B$ , then  $A$  would be a subconcept of  $B$ ;
  - ▶ if only  $B \rightarrow A$  holds then  $B$  should be a subconcept of  $A$ ;
  - ▶ we have  $A \rightarrow B$  and  $B \rightarrow A$ , that's why they come side by side in the lattice.
  - ▶ So attributes from a concept imply each other.

## Example

We have the next XML FDs:

$\langle C_{Orders}, ./OrderID, ./OrderDetails/OrderID \rangle$

$\langle C_{Orders}, ./OrderID, ./Orders@key \rangle$

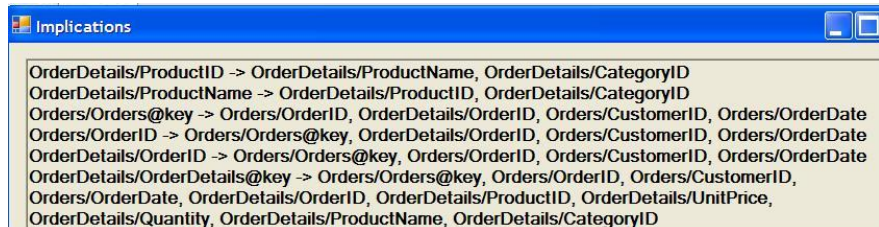
$\langle C_{Orders}, ./Orders@key, ./OrderID \rangle$

$\langle C_{Orders}, ./Orders@key, ./OrderDetails/OrderID \rangle$

$\langle C_{Orders}, ./OrderDetails/OrderID, ./Orders@key \rangle$

$\langle C_{Orders}, ./OrderDetails/OrderID, ./OrderID \rangle$

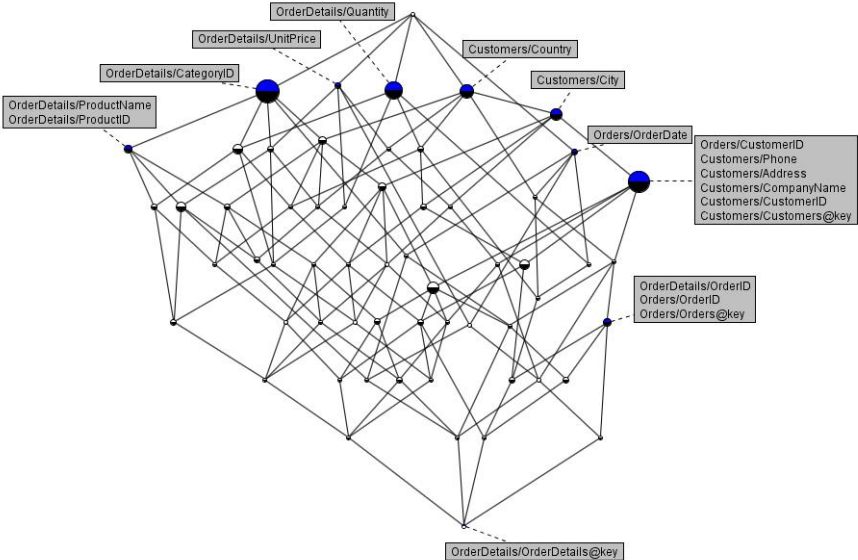
# The functional dependencies found by software FCAMineXFD



```
OrderDetails/ProductID -> OrderDetails/ProductName, OrderDetails/CategoryID
OrderDetails/ProductName -> OrderDetails/ProductID, OrderDetails/CategoryID
Orders/Orders@key -> Orders/OrderID, OrderDetails/OrderID, Orders/CustomerID, Orders/OrderDate
Orders/OrderID -> Orders/Orders@key, OrderDetails/OrderID, Orders/CustomerID, Orders/OrderDate
OrderDetails/OrderID -> Orders/Orders@key, Orders/OrderID, Orders/CustomerID, Orders/OrderDate
OrderDetails/OrderDetails@key -> Orders/Orders@key, Orders/OrderID, Orders/CustomerID,
Orders/OrderDate, OrderDetails/OrderID, OrderDetails/ProductID, OrderDetails/UnitPrice,
OrderDetails/Quantity, OrderDetails/ProductName, OrderDetails/CategoryID
```

Figure: Functional dependencies in tuple class  $C_{Orders}$

# The concept lattice for the whole XML document



- ▶ we can see the hierarchy of the analyzed data:
- ▶ the node labeled by *Customers/Country* is on a higher level, than node labeled by *Customers/City*;
- ▶ the Customer's node with every attribute is a subconcept of node labeled *Customers/City*;
- ▶ in our XML data, every customer has different name, address, phone number, so these attributes appear in one concept node and **imply each other**;
- ▶ the Orders node in XML is child of Customers, in the lattice, the node labeled with the key of Orders node, is subconcept of Customers node, so the hierarchy is visible;
- ▶ these are 1:n relationships, from Country to City, from City to Customers, from Customers to Orders.
- ▶ information about products is on the other side of the lattice; Products are in n:m relationship with Customers, linked by OrderDetail node in this case.

# FDs for the whole XML document

```
Implications
Customers/City -> Customers/Country
OrderDetails/ProductID -> OrderDetails/ProductName, OrderDetails/CategoryID
OrderDetails/ProductName -> OrderDetails/ProductID, OrderDetails/CategoryID
Customers/Customers@key -> Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/Phone, Orders/CustomerID, Customers/City, Customers/Country
Customers/CustomerID -> Customers/Customers@key, Customers/CompanyName, Customers/Address, Customers/Phone, Orders/CustomerID, Customers/City, Customers/Country
Customers/CompanyName -> Customers/Customers@key, Customers/CustomerID, Customers/Address, Customers/Phone, Orders/CustomerID, Customers/City, Customers/Country
Customers/Address -> Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Phone, Orders/CustomerID, Customers/City, Customers/Country
Customers/Phone -> Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Orders/CustomerID, Customers/City, Customers/Country
Orders/CustomerID -> Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/Phone, Customers/City, Customers/Country
Orders/Orders@key -> Orders/OrderID, OrderDetails/OrderID, Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/City, Customers/Country, Customers/Phone,
Orders/CustomerID, Orders/OrderDate
Orders/OrderID -> Orders/Orders@key, OrderDetails/OrderID, Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/City, Customers/Country, Customers/Phone,
Orders/CustomerID, Orders/OrderDate
OrderDetails/OrderID -> Orders/Orders@key, Orders/OrderID, Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/City, Customers/Country, Customers/Phone,
Orders/CustomerID, Orders/OrderDate
OrderDetails/OrderDetails@key -> Customers/Customers@key, Customers/CustomerID, Customers/CompanyName, Customers/Address, Customers/City, Customers/Country, Customers/Phone, Orders/Orders@key, Orders/OrderID,
Orders/CustomerID, Orders/OrderDate, OrderDetails/OrderID, OrderDetails/ProductID, OrderDetails/UnitPrice, OrderDetails/Quantity, OrderDetails/ProductName, OrderDetails/CategoryID
```

## Finding XML keys

FDs with RHS as  $./@key$  values can be used to detect the keys in XML.

In tuple class  $C_{Orders}$  we have XML FD:

- ▶  $\langle C_{Orders}, ./OrderID, ./@key \rangle$ , which implies that
  - ▶  $\langle C_{Orders}, ./OrderID \rangle$  is an XML key.
- ▶  $\langle C_{Orders}, ./OrderDetails/OrderID, ./@key \rangle$ , so
  - ▶  $\langle C_{Orders}, ./OrderDetails/OrderID \rangle$  is an XML key too.

In tuple class  $C_{Customers}$  software found XML FD:

- ▶  $\langle C_{Customers}, ./CustomerID, ./@key \rangle$ , which implies that
  - ▶  $\langle C_{Customers}, ./CustomerID \rangle$  is an XML key.
- ▶ other detected XML keys are:
  - ▶  $\langle C_{Customers}, ./Orders/CustomerID \rangle$ ;
  - ▶  $\langle C_{Customers}, ./CompanyName \rangle$ ;
  - ▶  $\langle C_{Customers}, ./Address \rangle$ ;
  - ▶  $\langle C_{Customers}, ./Phone \rangle$ .

## Detecting XML data redundancy

- ▶ having the set of functional dependencies for XML data in a tuple class, we can detect interesting functional dependencies.
- ▶ in essential tuple class  $C_{Orders}$  an interesting FD:  
 $\langle C_{Orders}, ./OrderDetails/ProductID, ./OrderDetails/ProductName \rangle$
- ▶ but  $\langle C_{Orders}, ./OrderDetails/ProductID \rangle$  is not an XML key
  - ▶ **So it is a data redundancy.**
- ▶ the same reason applies for XML FD  
 $\langle C_{Orders}, ./OrderDetails/ProductName, ./OrderDetails/ProductID \rangle$ .
- ▶ the other XML FD's have as LHS a key for tuple class  $C_{Orders}$ .



# Conclusions

- ▶ This paper introduces an approach for mining functional dependencies in XML documents based on FCA.
- ▶ Based on the flat representation of XML, we constructed the concept lattice.
- ▶ We analyzed the resulted concepts, which allowed us to discover a number of interesting dependencies.
- ▶ Our framework offers an graphical visualization for dependency exploration.

## Future Work

- ▶ given the set of dependencies discovered by our tool:
- ▶ propose a normalization algorithm for converting any XML schema into a correct one