
XML und Datenbanken

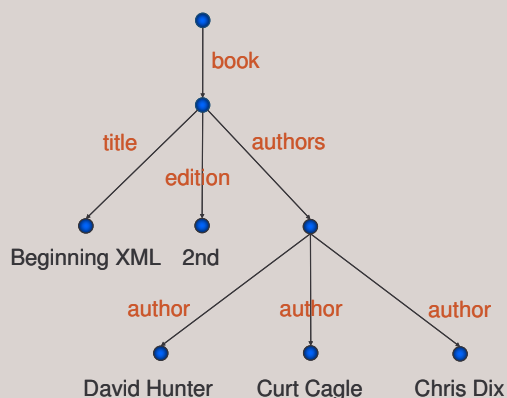
Übersicht

- XML persistent speichern: 3 Möglichkeiten
 - Exkurs: relationales Datenmodell
 - Abbildung relationales Datenmodell → XML
 - Datenmodellierung mit XML
- Schwerpunkt: Vergleich XML mit relationalem Datenmodell
 - vollständige Darstellung des Themas:
www.rpbouret.com/xml/XMLAndDatabases.htm

XML persistent speichern

XML vs. relationales Datenmodell

XML



- Hierarchie (Baum)
- Kind-Elemente: Reihenfolge relevant
- Anfragen: XPath

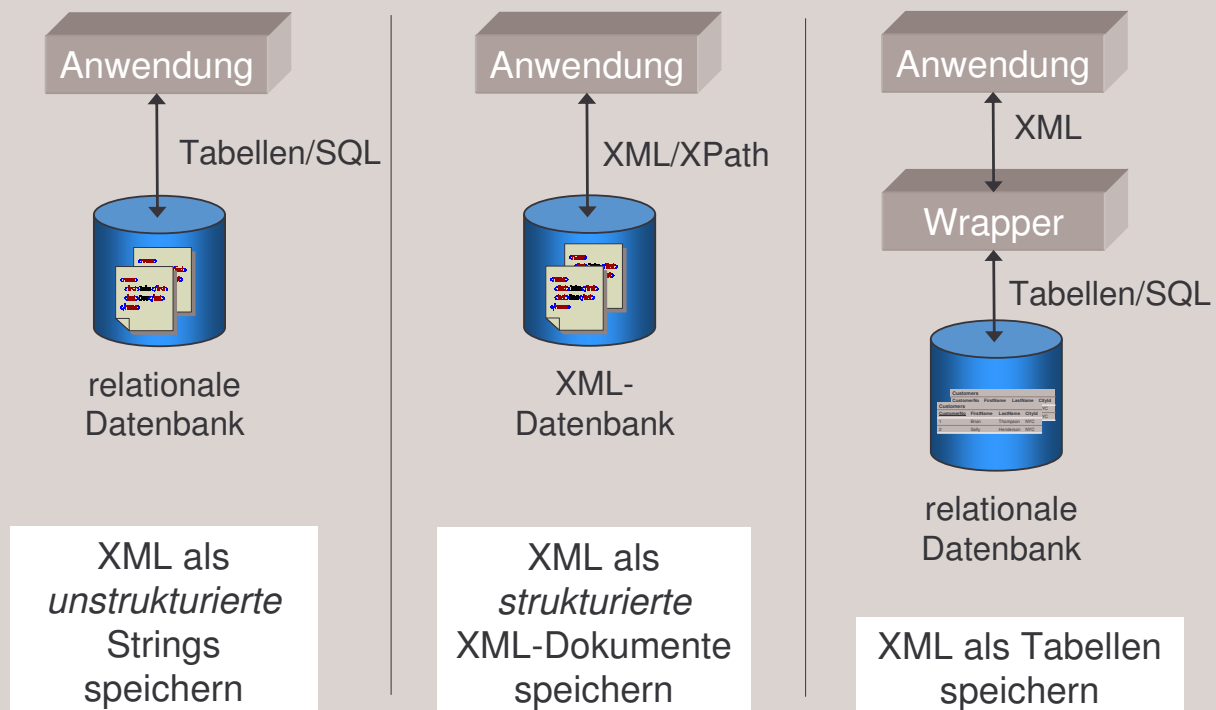
relationales Datenmodell

Book		
<u>BookKey</u>	Title	Edition
1	Beginning XML	2nd

Author		
<u>AuthorKey</u>	FirstName	LastName
1	David	Hunter
2	Kurt	Cagle
3	Chris	Dix

Authorship		
<u>Key</u>	BookKey	AuthorKey
1	1	1
2	1	2
3	1	3

- Tabellen, Schlüssel
- Spalten und Zeilen: Reihenfolge egal
- Anfragen: SQL

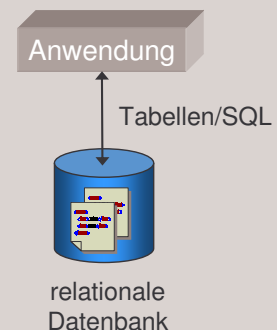


1. XML in Datenbank als String speichern

- hierarchische XML-Struktur als Wert eines Feldes gespeichert
- XML-Struktur wird als String serialisiert

Vor- und Nachteile

- + vorhandenes relationales Datenbanksystem (RDBMS) kann genutzt werden
- + triviale Schnittstelle zwischen XML und RDBMS
- keine komplexen Anfragen auf XML-Struktur



- **Datenbanksystem (DBMS):** Software, die es erlaubt, Daten persistent zu speichern
- **Datenbank (DB):** Datensatz, der in einem DBMS gespeichert ist

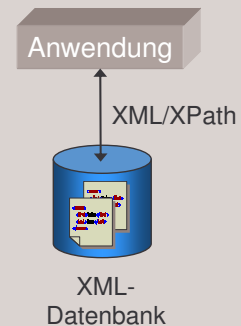
2. XML in XML-Datenbank speichern



- Übersicht XML-Datenbanken:
www.rpbouret.com/xml/XMLDatabaseProds.htm

Vor- und Nachteile

- + XML-Datenmodell direkt unterstützt
- + XPath wird unterstützt
- neue Datenbank nötig
- XML-Datenbanken noch nicht interoperabel
- schwierige Integration mit bestehenden relationalen Datenbanken (RDB)
- keine Systematik der Datenmodellierung



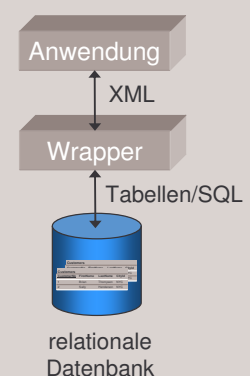
3. XML-Wrapper für relationale Datenbank



- RDMS mit XML-Schnittstelle
- Tabellen und Schlüssel in XML kodiert:
Kodierung einfach und *ohne* Informationsverlust
- Anfragen: SQL mit XML-Funktionen, z.B. SQL/XML

Vor- und Nachteile

- + vorhandenes RDBMS kann genutzt werden
- + von heutigen RDBMS unterstützt
- + einfache Abbildung RDB → XML
- + Systematik der Datenmodellierung
- Möglichkeiten der Datenmodellierung von XML nicht voll ausgeschöpft



- + vorhandenes RDBMS kann genutzt werden
- + triviale Schnittstelle XML↔RDBMS
- *keine* komplexen Anfragen auf XML-Daten möglich

XML als
unstrukturierte
Strings speichern

- + XML-Datenmodell direkt unterstützt
- neue Datenbank nötig
- XML-Datenbanken noch nicht interoperabel
- schwierige Integration mit bestehenden RDB
- **keine Systematik der Datenmodellierung**

XML als *strukturierte*
XML-Dokumente
speichern

- + vorhandenes RDBMS kann genutzt werden
- + von heutigen RDBMS unterstützt
- + **einfache Abbildung RDB → XML**
- + **Systematik der Datenmodellierung**
- **Möglichkeiten der Datenmodellierung von XML nicht voll ausgeschöpft**

XML als Tabellen
speichern

Exkurs: Relationales Datenmodell

Relationales Datenmodell



Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

- 1970 von Codd eingeführt
 - Daten in **Tabellen** organisiert
 - Tabelle repräsentiert n -stellige Beziehung (**Relation**) zwischen *primitiven* Daten.
- ⇒ keine geschachtelten Tabellen
- Tabelle besteht aus Spalten (**Felder**) und Zeilen (**Tupel**).
 - Zeilen und Spalten ungeordnet
 - Tabellen haben eindeutige Namen.

Primär- und Fremdschlüssel



- mindestens eine Spalte einer Tabelle ist **Primärschlüssel**: eindeutiger Repräsentant eines bestimmten Tupels
- bei mehreren Spalten: **zusammengesetzter Primärschlüssel**
- **Fremdschlüssel**: referenziert Primärschlüssel einer anderen Tabelle

Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Orders		
<u>OrderNo</u>	<u>CustomerNo</u>	ItemNo
121	1	FX100
5	1	FX200

Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Orders		
<u>OrderNo</u>	<u>CustomerNo</u>	ItemNo
121	1	FX100
5	1	FX200

- Primärschlüssel müssen für jedes Tupel einen Wert haben.
- Primärschlüssel müssen innerhalb der Tabelle eindeutig sein.
- Für jeden Fremdschlüssel muss ein zugehöriger Primärschlüssel existieren.

Typen von Beziehungen

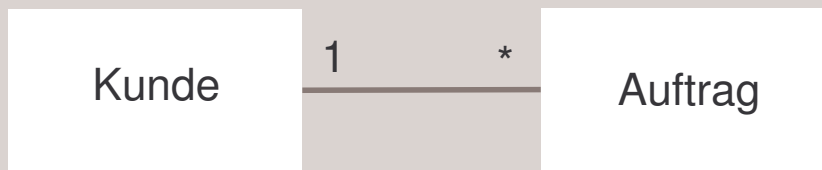
Tabellen (Relationen):

- Beziehungen zwischen primitiven Daten
- meist Objekte der realen Welt, wie z.B. Kunden oder Aufträge.
- Zwischen zwei Objekten der realen Welt kann eine 1:1-, 1:N- oder N:M-Beziehung bestehen.

Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Wie werden 1:1-, 1:N- und N:M-Beziehungen zwischen Tabellen ausgedrückt?

1:N-Beziehung



- Bestimmter Kunde kann mehrere Aufträge erteilen.
- Umgekehrt ist aber jedem Auftrag immer *genau ein* Kunde zugeordnet.
- Zwischen Kunden und Aufträgen besteht eine 1:N-Beziehung.

1:N-Beziehung im relationalen Modell



Primärschlüssel

1

Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

:

eindeutig

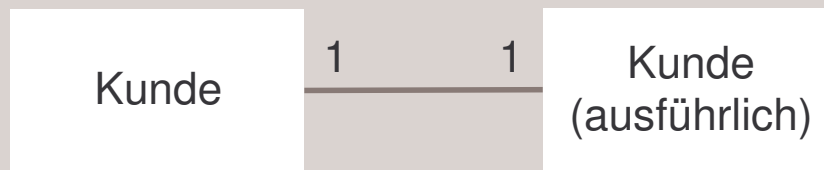
Fremdschlüssel

N

Orders		
<u>OrderNo</u>	<u>CustomerNo</u>	ItemNo
121	1	FX100
5	1	FX200

nicht eindeutig

1:1-Beziehung



- 1:1-Beziehungen eher selten
- Beispiel:
 - zwei unterschiedliche Kunden-Tabellen
 - kompakte Version für Außendienstmitarbeiter
 - ausführlichere Version für die interne Verwaltung
 - Zwischen den beiden Tabellen sollte eine 1:1-Beziehung bestehen.

1:1-Beziehung im relationalen Modell

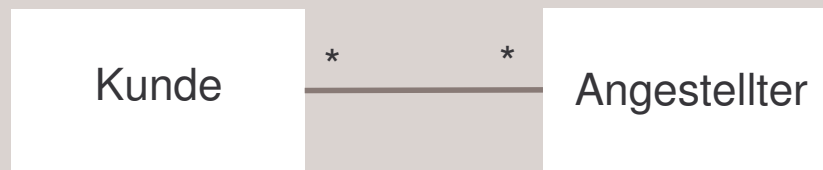


Customers			
<u>CustomerNo</u>	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Customers (detailed)					
<u>CustomerNo</u>	LastName	FirstName	CityId	CreditCard	CreditCardNo
1	Brian	Thompson	NYC	Amex	100900402344
2	Sally	Henderson	NYC	Visa	100800402e33

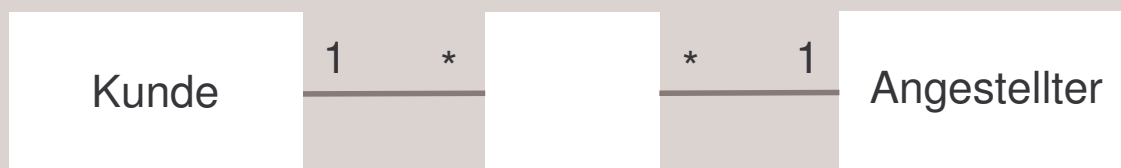
- beide Schlüssel gleichzeitig Primär- und Fremdschlüssel

N:M-Beziehung



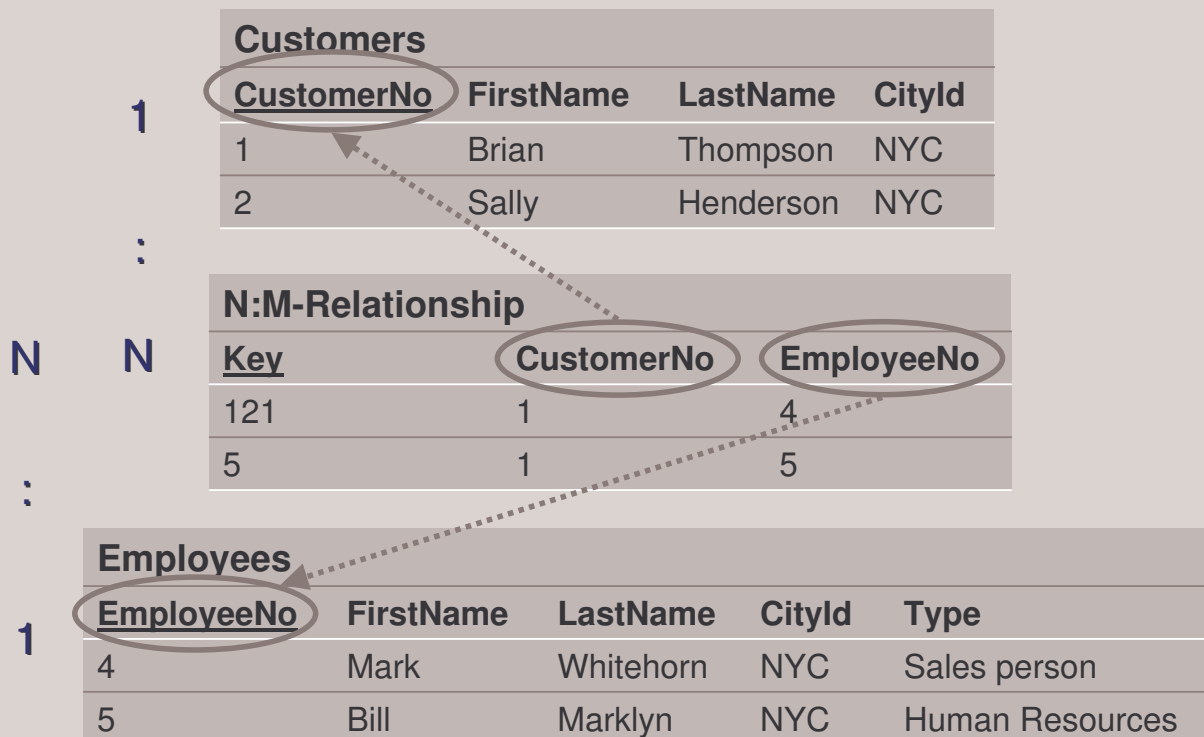
- Ein Angestellter kann mehrere Kunden betreuen.
- Umgekehrt kann ein Kunde gleichzeitig von mehreren Angestellten betreut werden.
- Zwischen Kunden und Angestellten besteht eine N:M-Beziehung.

N:M-Beziehung im relationalen Modell



- im relationalen Datenmodell N:M-Beziehung *nicht* direkt darstellbar
- muss in zwei 1:N-Beziehungen aufgebrochen werden
- Dritte Tabelle enthält Fremdschlüssel beider Tabellen.

N:M-Beziehung im relationalen Modell



Alternative Darstellung



- Hilfsmittel zur Datenmodellierung
- für relationales Datenmodell formal definiert

Ziele

- Eigenschaften (Felder) zu passenden Objekten (Tabellen) gruppieren
- Redundante Informationen eliminieren
- Sicherstellen, dass jede Information eindeutig identifiziert werden kann

Mittel

- Verständnis der Bedeutung der Daten
- Verständnis funktionaler Abhängigkeiten

- Beispiel: Fläche = Höhe x Breite
- Fläche **funktional abhängig** von der Höhe und Breite:
- Fläche = $f(\text{Höhe}, \text{Breite})$, für eine Funktion f
- es gibt auch funktionale Abhängigkeiten zwischen Feldern einer relationalen Datenbank

Beispiel



Orders					
<u>OrderNo</u>	<u>ItemNo</u>	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9

- Annahme: jedem Auftrag *genau ein* Angestellter (Vermittler) zugeordnet
 - ⇒ $\text{EmployeeNo} = f(\text{OrderNo})$
 - ⇒ EmployeeNo funktional abhängig von OrderNo.

Wichtig: Funktionale Abhängigkeiten nicht an Daten selbst zu erkennen, sondern daran, wie sie verwendet werden.

Weitere funktionale Abhängigkeiten



Orders					
<u>OrderNo</u>	<u>ItemNo</u>	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9

- $\text{Quantity} = f_1(\text{OrderNo}, \text{ItemNo})$
- $\text{ItemName} = f_2(\text{ItemNo})$
- $\text{CustomerNo} = f_3(\text{OrderNo})$

- verschiedene Stufen, die aufeinander aufbauen:
1., 2., 3. Normalform (\Rightarrow Anhang)

Grundidee

- Unterscheidung funktionaler Abhängigkeiten in notwendige und nicht erlaubte
- Sei P der Primärschlüssel einer Tabelle.
- Seien F_i ein beliebiges Nicht-Schlüssel-Feld der Tabelle.

notwendig


- $F_i = f(P)$

nicht erlaubt

- $F_i = f(P')$, P' echte Teilmenge von P
- $F_i = f(P)$, jedoch auch $F_i = f(F_j)$, $F_j = f(P)$ für ein weiteres Nicht-Schlüssel-Feld F_j

Beispiel

Orders					
<u>OrderNo</u>	<u>ItemNo</u>	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9




notwendig

- $F_i = f(\text{OrderNo}, \text{ItemNo})$, $F_i = \text{EmployeeNo}, \dots, \text{Quantity}$

nicht erlaubt

- $\text{ItemName} = f(\text{ItemNo})$
- $\text{CustomerNo} = f(\text{OrderNo})$
- $\text{EmployeeNo} = f(\text{OrderNo})$

Customers				
<u>CustomerNo</u>	FirstName	LastName	CityId	City
1	Brian	Thompson	NYC	New York City
2	Sally	Henderson	NYC	New York City



notwendig

- $F_i = f(\text{CustomerNo}), F_i = \text{FirstName}, \dots, \text{CityId}, \text{City}$

nicht erlaubt

- $\text{City} = f(\text{CityId})$

Abbildung relationales Datenmodell → XML

- Relationales Datenmodell kann einfach und ohne Informationsverlust in XML kodiert werden.
- Kodierung könnte als Standard für RDBMS dienen.
- Hierfür gibt es allerdings *keinen* etablierten Standard.
- inoffizieller Vorschlag:
<http://www.w3.org/XML/RDB.html>

Kodierung einer RDB in XML

```
<Database>
  <EmployeeTable>
    <EmployeeTuple>
      <EmployeeNo>k1</EmployeeNo>
      <FirstName>Mark</FirstName>
      <LastName>Whitehorn</LastName>
      <CityId>NYC</CityId>
      <Type>Sales Person</Type>
    </EmployeeTuple>
    <EmployeeTuple>
      <EmployeeNo>k2</EmployeeNo>
      <FirstName>Bill</FirstName>
      <LastName>Marklyn</LastName>
      <CityId>NYC</CityId>
      <Type>Human Resources</Type>
    </EmployeeTuple>
  </EmployeeTable>
  ...
```

- Wurzel-Element = Name der Datenbank
- für jede Tabelle genau ein Kind-Element
- darunter für jedes Tupel genau ein Kind-Element
- darunter für jede Spalte ein Kind-Element

Kodierung von Null Values

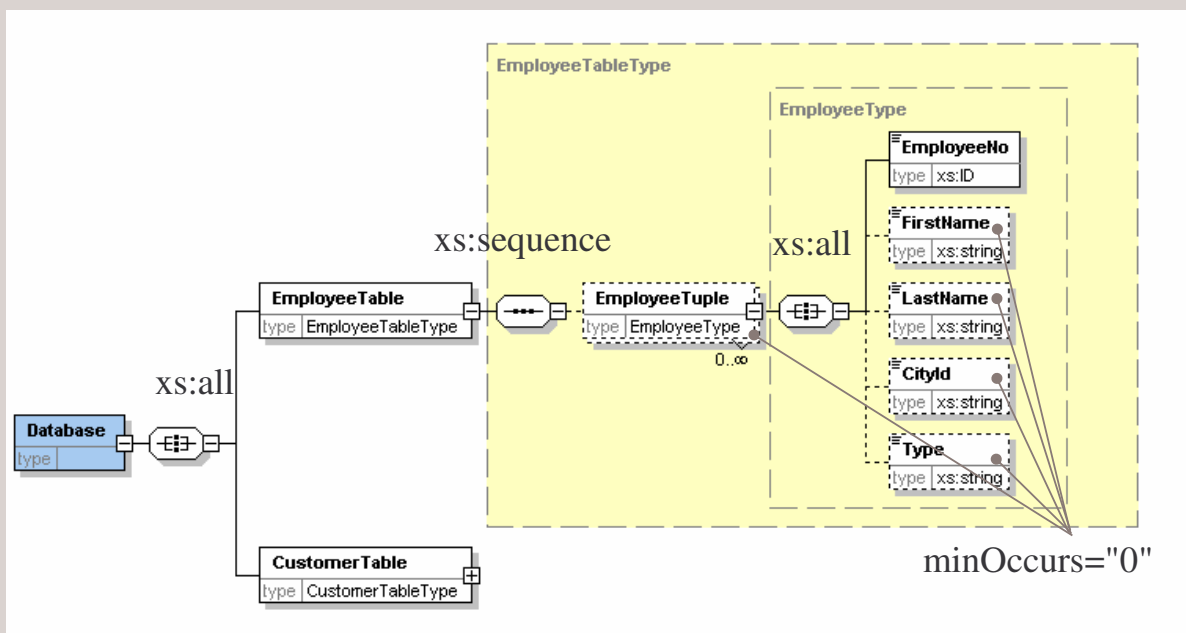


```

<Database>
  <EmployeeTable>
    <EmployeeTuple>
      <EmployeeNo>k1</EmployeeNo>
      <FirstName>Mark</FirstName>
      <LastName>Whitehorn</LastName>
      <CityId>NYC</CityId>
      <Type>Sales Person</Type>
    </EmployeeTuple>
    <EmployeeTuple>
      <EmployeeNo>k2</EmployeeNo>
      <FirstName>Bill</FirstName>
      <LastName>Marklyn</LastName>
      <Type></Type>
    </EmployeeTuple>
  </EmployeeTable>
  ...
  
```

- **Leerwerte (null values):**
undefinierte Werte
- Kodierung:
entsprechendes Element (= Feld)
einfach weglassen
- dadurch
Unterscheidung
zu leerem Inhalt

Das zugehörige XML-Schema



- Reihenfolge der Tabellen, Tupel und Spalten egal

Kodierung von Schlüssel



- Primärschlüssel vom Typ `xs:ID`.
- Fremdschlüssel vom Typ `xs:IDREF`.

Probleme dieser Kodierung

- keine *zusammengesetzten* Primärschlüssel darstellbar
- Statt Eindeutigkeit innerhalb der Tabelle, erzwingt `xs:ID` Eindeutigkeit aller Attribute mit Typ `xs:ID`
- ⇒ Zwei Tabellen dürfen keine identischen Primärschlüssel haben.
- Statt eines ganz bestimmten Primärschlüssels, referenziert `xs:IDREF` *beliebigen* Primärschlüssel mit Typ `xs:ID`.

Beispiel



```
<Database>
  <EmployeeTable>
    <EmployeeTuple>
      <EmployeeNo>ID4</EmployeeNo>
      <FirstName>String</FirstName>
      <LastName>String</LastName>
      <CityId>ID5</CityId>
      <Type>String</Type>
    </EmployeeTuple>
  </EmployeeTable>
  <CustomerTable>
    <CustomerTuple>
      <CustomerNo>ID5</CustomerNo>
      ...
    </CustomerTuple>
  </CustomerTable>
</Database>
```

Primärschlüssel müssen in *gesamter* Datenbank (nicht nur in Tabelle) eindeutig sein.

Fremdschlüssel kann sich auf *beliebigen* Primärschlüssel (nicht nur auf CityID) beziehen.

Kodierung von Primärschlüssel mit key



```
<xs:element name="Database">
  <xs:complexType>
    Definition der Tabellen
  </xs:complexType>
  <xs:key name="EmployeeKey">
    <xs:selector xpath="EmployeeTable/EmployeeTuple"/>
    <xs:field xpath="EmployeeNo"/>
  </xs:key>
</xs:element>
```

⇒ EmployeeNo innerhalb EmployeeTable eindeutig

- **name**: eindeutiger Namen des Primärschlüssels
- **xs:selector**: spezifiziert Kontext, auf die die Eindeutigkeitsbedingung angewandt wird.
- ein oder mehrere **xs:field**-Elemente: Elemente/Attribute, die zusammen eindeutig sein sollen.

Kodierung von Fremdschlüssel mit keyref



```
<xs:element name="Database">
  <xs:complexType>
    Definition der Tabellen
  </xs:complexType>
  <xs:keyref name="CityIDKeyRef" refer="CityIDKey">
    <xs:selector xpath="*/*/"/>
    <xs:field xpath="CityID"/>
  </xs:keyref>
</xs:element>
```

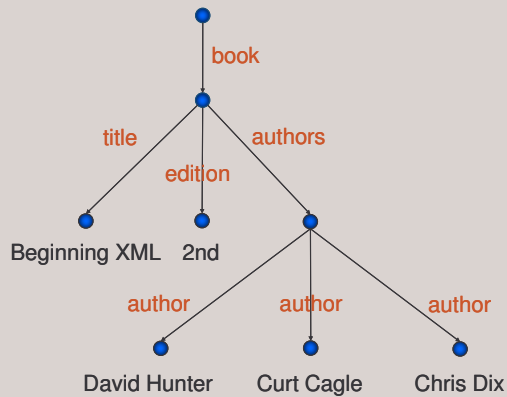
⇒ Wert von CityId immer definiert

- **refer**: Auf welchen Primärschlüssel wird verwiesen?
- **xs:selector**: Wo ist der Fremdschlüssel zu finden?
- **xs:field**: Aus welchen Feldern besteht der Fremdschlüssel?

- Abbildung relationale Datenbank → XML ohne Informationsverlust möglich
- gilt auch für Primär- und Fremdschlüssel
- Instanz kodiert Datenbank
- XML-Schema kodiert Datenbankschema

Datenmodellierung mit XML

XML



relationales Datenmodell

Book		
BookKey	Title	Edition
1	Beginning XML	2nd

Author		
AuthorKey	FirstName	LastName
1	David	Hunter
2	Kurt	Cagle
3	Chris	Dix

Authorship		
Key	BookKey	AuthorKey
1	1	1
2	1	2
3	1	3

XML zur Datenmodellierung

- XML flexibler als relationales Datenmodell:

Relationales Datenmodell

- keine geschachtelten Tabellen
- N:M-Beziehungen müssen in eigenen Tabellen ausgelagert werden.

XML

- erlaubt geschachtelte Strukturen
- N:M-Beziehungen können unkompliziert ausgedrückt werden.

Warum also nicht die hierarchischen Strukturierungsmöglichkeiten von XML voll ausnutzen?

```
<Employee>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <City>
    <Name>New York City</Name>
    <CityId>NYC</CityId>
  </City>
  <Type>Sales Person</Type>
  <Orders>
    <Order>
      <OrderNo>121</OrderNo>
      <OrderItems>...</OrderItems>
      <CustomerNo>999</CustomerNo>
    </Order>
  </Orders>
</Employee>
```

© Klaus Schild 2005

- Könnte so zwischen Vertriebs- und Gehaltsabteilung ausgetauscht werden
- als Austauschformat OK
- auch als Speicherformat OK?

43

Löschanomalie

```
<Employee>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <City>
    <Name>New York City</Name>
    <CityId>NYC</CityId>
  </City>
  <Type>Sales Person</Type>
  <Orders>
    <Order>
      <OrderNo>121</OrderNo>
      <OrderItems>...</OrderItems>
      <CustomerNo>999</CustomerNo>
    </Order>
  </Orders>
</Employee>
```

© Klaus Schild 2005

- Wird ein Angestellten-Datensatz gelöscht, dann werden auch *alle* Aufträge gelöscht, die er vermittelt hat.
 - Gefahr, ungewollt Informationen zu löschen
- ⇒ Order-Informationen auslagern und durch Fremdschlüssel OrderNo ersetzen.

44

```
<Employee>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <City>
    <Name>New York City</Name>
    <CityId>NYC</CityId>
  </City>
  <Type>Sales Person</Type>
  <Orders>
    <OrderNo>121</OrderNo>
  </Orders>
</Employee>
```

```
<Order-DB>
  <Order>
    <OrderNo>121</OrderNo>
    <OrderItems>...</OrderItems>
    <CustomerNo>999</CustomerNo>
  </Order>
</Order-DB>
```

Änderungsanomalie

```
<Employee>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <City>
    <Name>New York City</Name>
    <CityId>NYC</CityId>
  </City>
  <Type>Sales Person</Type>
  <Orders>
    <OrderNo>121</OrderNo>
  </Orders>
</Employee>
```

- Wird CityId oder Name geändert, dann muss diese Änderung in *allen* Angestellten-Datensätzen nachvollzogen werden.
 - unnötiger Verwaltungsaufwand
 - Gefahr von Inkonsistenzen
- ⇒ City-Informationen auslagern und hier durch Fremdschlüssel ersetzen.

```
<Employee-DB>
  <Employee>
    <EmployeeNo>4</EmployeeNo>
    <Name>
      <First>Mark</First>
      <Last>Whitehorn</Last>
    </Name>
    <CityKey>C123</CityKey>
    <Type>Sales Person</Type>
    <Orders>
      <OrderNo>121</OrderNo>
    </Orders>
  </Employee>
</Employee-DB>
```

```
<Order-DB>
  <Order>
    <OrderNo>121</OrderNo>
    <OrderItems>...</OrderItems>
    <CustomerNo>999</CustomerNo>
  </Order>
</Order-DB>
```

```
<City-DB>
  <City>
    <CityKey>C123</CityKey>
    <CityId>NYC</CityId>
    <Name>New York City</Name>
  </City>
</City-DB>
```

Noch eine Änderungsanomalie

```
<Order-DB>
  <Order>
    <OrderNo>121</OrderNo>
    <OrderItems>
      <OrderItem>
        <ItemNo>FX100</ItemNo>
        <ItemName>Black-Bag</ItemName>
        <Quantity>1000</Quantity>
      </OrderItem>
    </OrderItems>
    <CustomerNo>999</CustomerNo>
  </Order>
</Order-DB>
```

- Wird ItemName geändert, dann muss diese Änderung in *allen* Order-Datensätzen nachvollzogen werden.
- ⇒ Zuordnung ItemNo/ItemName auslagern und durch Fremdschlüssel ItemNo ersetzen.

Verbesserte Modellierung



```
<Order-DB>
  <Order>
    <OrderNo>121</OrderNo>
    <OrderItems>
      <OrderItem>
        <ItemNo>FX100</ItemNo>
        <Quantity>1000</Quantity>
      </OrderItem>
    </OrderItems>
    <CustomerNo>999</CustomerNo>
  </Order>
</Order-DB>
```

```
<Inventory-DB>
  <Item>
    <ItemNo>FX100</ItemNo>
    <ItemName>Black-Bag</ItemName>
  </Item>
</Inventory-DB>
```

Anfrageoptimierung



```
<Order-DB>
  <Order>
    <OrderNo>121</OrderNo>
    <OrderItems>
      <OrderItem>
        <ItemNo>FX100</ItemNo>
        <Quantity>1000</Quantity>
      </OrderItem>
    </OrderItems>
    <CustomerNo>999</CustomerNo>
  </Order>
</Order-DB>
```

Wer ist Vermittler?

- Hier fehlt Verweis auf Vermittler (EmployeeNo).
- Vermittler normalerweise Ansprechpartner für Kundenauftrag

⇒ Angestellten-Datenbank muss durchsucht werden, um Vermittler eines Auftrages zu ermitteln.

```
<Order-DB>
```

```
<Order>
```

```
<OrderNo>121</OrderNo>
```

```
<OrderItems>
```

```
<OrderItem>
```

```
<ItemNo>FX100</ItemNo>
```

```
<Quantity>1000</Quantity>
```

```
</OrderItem>
```

```
</OrderItems>
```

```
<CustomerNo>999</CustomerNo>
```

```
<EmployeeNo>4</EmployeeNo>
```

```
</Order>
```

```
</Order-DB>
```

```
<Employee-DB>
```

```
<Employee>
```

```
<EmployeeNo>4</EmployeeNo>
```

```
<Name>
```

```
<First>Mark</First>
```

```
<Last>Whitehorn</Last>
```

```
</Name>
```

```
<CityKey>C123</CityKey>
```

```
<Type>Sales Person</Type>
```

```
<Orders>
```

```
<OrderNo>121</OrderNo>
```

```
</Orders>
```

```
</Employee>
```

```
</Employee-DB>
```

statt Verweis Angestellter → Auftrag:
Verweis Auftrag → Vermittler

Endgültige Modellierung

```
<Order>
```

```
<OrderNo>121</OrderNo>
```

```
<OrderItems>
```

```
<OrderItem>
```

```
<ItemNo>FX100</ItemNo>
```

```
<Quantity>1000</Quantity>
```

```
</OrderItem>
```

```
</OrderItems>
```

```
<CustomerNo>999</CustomerNo>
```

```
<EmployeeNo>4</EmployeeNo>
```

```
</Order>
```

```
<Item>
```

```
<ItemNo>FX100</ItemNo>
```

```
<ItemName>Black-Bag</ItemName>
```

```
</Item>
```

```
<Employee>
```

```
<EmployeeNo>4</EmployeeNo>
```

```
<Name>
```

```
<First>Mark</First>
```

```
<Last>Whitehorn</Last>
```

```
</Name>
```

```
<CityKey>C123</CityKey>
```

```
<Type>Sales Person</Type>
```

```
</Employee>
```

```
<City>
```

```
<CityKey>C123</CityKey>
```

```
<CityId>NYC</CityId>
```

```
<Name>New York City</Name>
```

```
</City>
```

Vergleich mit relationalem Datenmodell



```
<OrderTuple>
  <OrderNo>121</OrderNo>
  <OrderItemTable>
    <OrderItemTuple>
      <ItemNo>FX100</ItemNo>
      <Quantity>1000</Quantity>
    </OrderItemTuple>
  </OrderItemTable>
  <CustomerNo>999</CustomerNo>
  <EmployeeNo>4</EmployeeNo>
</OrderTuple>
```

```
<ItemTuple>
  <ItemNo>FX100</ItemNo>
  <ItemName>Black-Bag</ItemName>
</ItemTuple> ✓
```

```
<EmployeeTuple>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <CityKey>NYC</CityKey>
  <Type>Sales Person</Type>
</EmployeeTuple>
```

```
<CityTuple>
  <CityKey>C123</CityKey>
  <CityId>NYC</CityId>
  <Name>New York City</Name>
</CityTuple> ✓
```

Vergleich mit relationalem Datenmodell



```
<OrderTuple>
  <OrderNo>121</OrderNo>
  <OrderItemTable> N:M
    <OrderItemTuple>
      <ItemNo>FX100</ItemNo>
      <Quantity>1000</Quantity>
    </OrderItemTuple>
  </OrderItemTable>
  <CustomerNo>999</CustomerNo>
  <EmployeeNo>4</EmployeeNo>
</OrderTuple>
```

```
<ItemTuple>
  <ItemNo>FX100</ItemNo>
  <ItemName>Black-Bag</ItemName>
</ItemTuple> ✓
```

```
<EmployeeTuple>
  <EmployeeNo>4</EmployeeNo>
  <FirstName>Mark</FirstName>
  <LastName>Whitehorn</LastName>
  <CityKey>C123</CityKey>
  <Type>Sales Person</Type>
</EmployeeTuple> ✓
```

```
<CityTuple>
  <CityKey>C123</CityKey>
  <CityId>NYC</CityId>
  <Name>New York City</Name>
</CityTuple> ✓
```

```
<OrderSpecTuple>
  <OrderNo>121</OrderNo>
  <ItemNo>FX100</ItemNo>
  <Quantity>1000</Quantity>
</OrderSpecTuple>
```



```
<OrderTuple>
  <OrderNo>121</OrderNo>
  <CustomerNo>999</CustomerNo>
  <EmployeeNo>4</EmployeeNo>
</OrderTuple>
```



```
<ItemTuple>
  <ItemNo>FX100</ItemNo>
  <ItemName>Black-Bag</ItemName>
</ItemTuple>
```



```
<EmployeeTuple>
  <EmployeeNo>4</EmployeeNo>
  <First>Mark</First>
  <Last>Whitehorn</Last>
  <CityKey>C123</CityKey>
  <Type>Sales Person</Type>
</EmployeeTuple>
```



```
<CityTuple>
  <CityKey>C123</CityKey>
  <CityId>NYC</CityId>
  <Name>New York City</Name>
</CityTuple>
```



N:M-Beziehung als Relation
OrderSpec mit
zusammengesetztem
Primärschlüssel

Fazit aus dem Beispiel



Ausgangspunkt

- strukturiertes XML-Dokument als Speicherformat

Transformationen

- Beseitigung von Lösch- und Änderungsanomalien

Ergebnis

- mehrere, wesentlich flachere XML-Strukturen
- relationalem Datenmodell (in 3. NF) sehr ähnlich
- Ausnahme: N:M-Beziehungen als geschachtelte Strukturen

relationales Datenmodell

- schrittweise Normalformbildung

XML

- Normalformen aus relationalem Datenmodell *nicht* auf XML übertragbar
- Grund: relationales Datenmodell erlaubt keine geschachtelten Tabellen
- bisher *keine* formale Systematik der Datenmodellierung
- informelles Verfahren:
Asset-Oriented Modeling (Daum & Merten, System Architecture with XML, 2003).

Fazit insgesamt

- Für Daten mit vielen funktionalen Abhängigkeiten besser gleich professionelle Speicherformate (wie relationale Datenbanken) wählen.
- Für Text-Dokumente ohne funktionale Abhängigkeiten kann XML als Speicherformat benutzt werden.

Wie geht es weiter?



heutige Vorlesung

- ☑ XML persistent speichern: 3 Möglichkeiten
- ☑ Abbildung relationales Datenmodell → XML
- ☑ Datenmodellierung mit XML

heutiges Tutorium (2. Gruppe)

- Beispiel XSLT: XML ⇔ XML
- Beispiel XSLT: XML ⇔ XHTML-Layout

Vorlesung nächste Woche

- Web Services (insgesamt 4 Termine)



Anhang

1. Normalform (NF)



- Eine relationale Datenbank ist in **1. Normalform**, falls alle Tabellen
 - 1) einen Primärschlüssel haben und
 - 2) nur primitive Daten enthalten.
- Entspricht der Definition des relationalen Modells

2. Normalform (NF)



- Eine relationale Datenbank ist in **2. Normalform**, falls
 - 1) sie in 1. Normalform ist,
 - 2) alle Nicht-Schlüssel-Felder vom Primärschlüssel funktional abhängig sind und
 - 3) kein Nicht-Schlüssel-Feld bereits von einem Teil des Primärschlüssels funktional abhängig ist.

Orders					
<u>OrderNo</u>	<u>ItemNo</u>	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9

nicht in 2. NF

3. Normalform (NF)



- Eine relationale Datenbank ist in **3. Normalform**, falls
 - 1) sie in 2. Normalform ist und
 - 2) alle Nicht-Schlüssel-Felder direkt (d.h. nicht transitiv) vom Primärschlüssel funktional abhängig sind.

Customers

<u>CustomerNo</u>	FirstName	LastName	CityId	City
1	Brian	Thompson	NYC	New York City
2	Sally	Henderson	NYC	New York City

nicht in 3. NF