XML und Datenbanken

Lernziele

- Wie kann eine relationale Datenbank in XML repräsentiert werden?
- Was sollte beachtet werden, wenn XML zur Datenspeicherung verwendet wird?

Relationales Datenmodell

Relationales Datenmodell

- 1970 von Codd eingeführt
- Daten werden in *Tabellen* organisiert.
- Tabelle repräsentiert *n*-stellige Beziehung *(Relation)* zwischen primitiven Daten.
- Tabelle besteht aus *Spalten (Felder)* und *Zeilen (Tupel)*.
- Zeilen und Spalten sind ungeordnet.

Customers

CustomerNo	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Relationales Datenmodell

- Tabellen haben einen eindeutigen Namen.
- Spalten haben einen innerhalb der Tabelle eindeutigen Namen.
- mindestens eine Spalte einer Tabelle als *Primärschlüssel* ausgezeichnet (bei mehreren Spalten: *zusammengesetzter Primärschlüssel*)
- Primärschlüssel innerhalb einer Tabelle eindeutig
- Referenziert eine Tabelle Primärschlüssel einer anderen Tabelle, so spricht man von einem *Fremdschlüssel*.

Primär- und Fremdschlüssel

Customers

CustomerNo	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Orders

OrderNo	CustomerNo	ItemNo
121	1	FX100
5	1	FX200

- Primärschlüssel müssen existieren und innerhalb der Tabelle eindeutig sein.
- Für jeden Fremdschlüssel muss ein zugehöriger Primärschlüssel existieren.

Typen von Beziehungen

- Mit Tabellen (Relationen) lassen sich Beziehungen zwischen primitiven Daten ausdrücken.
- Tabellen repräsentieren meist Objekte der realen Welt, wie z.B. Kunden oder Aufträge.
- Zwischen Objekte der realen Welt können unterschiedliche Typen von Beziehungen bestehen, wie 1:N- oder N:M-Beziehungen.

Wie werden 1:N- und N:M-Beziehungen ausgedrückt?

1:N-Beziehung

- Ein 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

CustomerNo	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

:

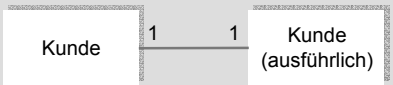
N

Orders

OrderNo	CustomerNo	ItemNo
121	1	FX100
5	1	FX200

Fremdschlüssel

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

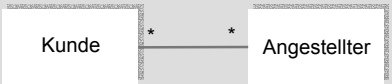
CustomerNo	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

Customers (details)

CustomerNo	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



- Bestimmter 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-Bezeiehung *nicht* direkt darstellbar
- muss in zwei 1:N-Beziehungen aufgebrochen werden
- Dritte Tabelle enthält Fremdschlüssel von beiden Tabellen.

N:M-Beziehung im relationalen Modell

Customers

CustomerNo	FirstName	LastName	CityId
1	Brian	Thompson	NYC
2	Sally	Henderson	NYC

N:M-Relationship

Key	CustomerNo	EmployeeNo
121	1	4
5	1	5

Employees

EmployeeNo	FirstName	LastName	CityId	Type
4	Mark	Whitehorn	NYC	Sales person
5	Bill	Marklyn	NYC	Human Resources

Kodierung des relationalen Modells in XML

Relationales Modell in XML

- Relationales Datenmodell kann auf einfache Weise in XML kodiert werden.
- Solche Kodierung könnte als Standardschnittelle für relationale Datenbanken dienen.
- Hierfür gibt allerdings *keinen* etablierten Standard.
- inoffizieller Vorschlag für eine Kodierung unter http://www.w3.org/XML/RDB.html

Kodierung 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

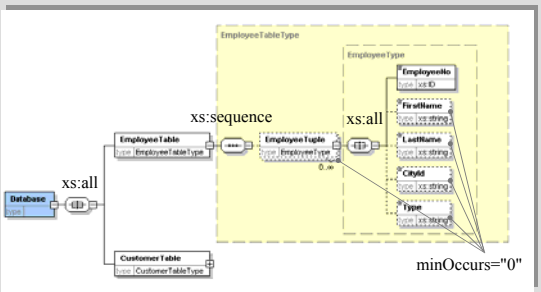
Kodierung von Null Values

```
<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 (engl. *null values*) sind 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 haben Typ xs:ID.
- Fremdschlüssel haben Typ xs:IDREF.

Probleme:

- mit xs:ID keine *zusammengesetzten* Primärschlüssel darstellbar
- Statt Eindeutigkeit innerhalb der Tabelle, erzwingt xs:ID Eindeutigkeit im *gesamten Dokument*.
- Zwei Tabellen dürfen also niemals einen identischen Primärschlüssel haben.
- Statt eines ganz bestimmten Primärschlüssels, referenziert xs:IDREF einen *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>
              <EmployeeNo>ID5</EmployeeNo>
              …
        </CustomerTuple>
  </CustomerTable>
</Database>
```

Primärschlüssel müssen im *gesamten* Dokument eindeutig sein.

Fremdschlüssel kann sich auf einen *beliebigen* Primärschlüssel 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>
```

- name gibt dem Primärschlüssel einen eindeutigen Namen.
- xs:selector spezifiziert den Kontext, auf die das Schlüssel-Constraint angewandt werden soll.
- Ein oder mehrere xs:field-Elemente geben die Elemente/Attribute an, 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>
```

*scheint mit xmlspy 5.2 nicht zu funktionieren

- name gibt dem Fremdschlüssel einen eindeutigen Namen.
- refer gibt den referenzierten Primärschlüssel an.
- Für die mit xs:selector und xs:field spezifizierten Elemente/Attribute (hier *Tabelle*/*Tupel*/CityID) muss Primärschlüssel mit passendem Typ existieren.*

Kodierung des relationalen Modells

Fazit:

- genaue Kodierung des relationalen Modells in XML möglich, einschl. Primär- und Fremdschlüssel
- passende Anfrage-Sprache:
  - XQuery
  - Working Draft von 2003
  - http://www.w3.org/TR/xquery/

XML als Speicherformat

XML zur Datenmodellierung

- Relationales Modell kann einfach in XML kodiert werden.
- Im Vergleich zum relationalen Model ist XML aber wesentlich flexibler:
- Relationales Modell erlaubt in Tabellen nur *primitive* Daten, geschachtelte Tabellen sind *nicht* erlaubt.
- XML erlaubt aber solche geschachtelten Strukturen.

Warum nicht die hierarchische Strukturierungsmöglichkeiten von XML voll ausnutzen?

XML zur Datenmodellierung

```
<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>…</OrderIntems>
      <CustomerNo>999</CustomerNo>
    </Order>
  </Orders>
</Employee>
```

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

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>
```

- Wird ein Angestellter gelöscht, dann werden auch *alle* Aufträge gelöscht, die der Angestellte vermittelt hat.
- Daher Order durch Fremdschlüssel OrderNo ersetzen (Referenz).

Verbesserte Modellierung

```
<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 geändert, dann muss diese Änderung in *allen* Angestellten-Datensätzen nachvollzogen werden.
- Daher City durch Fremdschlüssel CityId ersetzen (Referenz).

Verbesserte Modellierung

```
<Employee-DB>
 <Employee>
  <EmployeeNo>4</EmployeeNo>
  <Name>
    <First>Mark</First>
    <Last>Whitehorn</Last>
  </Name>
  <CityId>NYC</CityId>
  <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>
  <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.
- Deshalb Zuordnung ItemNo/ItemName gesondert abspeichern.
- Statt ItemNo und ItemName nur Fremdschlüssel ItemNo verwenden.

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?

- In Order fehlt der Verweis auf den Vermittler (EmployeeNo).
- Vermittler aber normalerweise Ansprechpartner für Kundenauftrag
- Ohne diesen Verweis muss die Angestellten-Datenbank durchsucht werden, um Vermittler eines Auftrages zu ermitteln.

Anfrageoptimierung

```
<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>
  <CityId>NYC</CityId>
  <Type>Sales Person</Type>
  <Orders>
    <OrderNo>121</OrderNo>
  </Orders>
 </Employee>
</Employee-DB>
```

statt Verweis in Angestellter zu Auftrag:
Verweis in Auftrag zu 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>
   <CityId>NYC</CityId>
   <Type>Sales Person</Type>
</Employee>
```

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

Vergleich mit relationalem Modell

```
<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>
  <CityId>NYC</CityId>
  <Type>Sales Person</Type>
</EmployeeTuple>
```

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

im relationalem Modell *keine* geschachtelten Strukturen

Relationale Modellierung

```
<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>
  <CityId>NYC</CityId>
  <Type>Sales Person</Type>
</EmployeeTuple>
```

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

N:M-Beziehung zwischen Order und Item muss in eigener Relation kodiert werden!

Relationale Modellierung

```
<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>
  <CityId>NYC</CityId>
  <Type>Sales Person</Type>
</EmployeeTuple>
```

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

N:M-Beziehung als Relation OrderSpec mit zusammengesetztem Primärschlüssel

Datenmodellierung

Datenmodellierung

- Problem: Welche Daten in welche Tabellen ablegen?
- Anforderungen:
  - Daten einfach zu Warten
  - relevante Anfragen effizient zu beantworten
- Faustregel: reale Objekte (wie Kunden und Aufträgen) in eigener Tabelle
- Diese einfache Faustregel meist nicht hinreichend, um obigen Anforderungen zu genügen.

Normalformen

- Normalformen: formale Gütegrade eines relationalen Datenmodells
- Ziele:
  - Eigenschaften zu passenden Objekten gruppieren
  - Redundante Informationen eliminieren
  - Sicherstellen, dass jede Information eindeutig identifiziert werden kann
- Mittel:
  - Verständnis der Bedeutung der Daten
  - Verständnis der funktionalen Abhängigkeiten zwischen Feldern

Funktionale Abhängigkeiten

- Beispiel: Fläche = Höhe x Breite
- Fläche *funktional abhängig* von der Höhe und Breite
- D.h. die Höhe zusammen mit der Breite bestimmen eindeutig die Fläche.
- Im relationalem Modell sind funktionale Abhängigkeiten *zwischen Feldern* relevant.

Beispiel

Orders

OrderNo	ItemNo	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
- EmployeeNo funktional abhängig von OrderNo.

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

Beispiel

Orders

OrderNo	ItemNo	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9

- Quantity vom gesamten Primärschlüssel abhängig, andere Felder nur von Teilen des Primärschlüssels.

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.

nicht in 2. NF

Orders

OrderNo	ItemNo	EmployeeNo	CustomerNo	ItemName	Quantity
121	3	4	1024	Nut	3
121	4	4	1024	Bolt	67
122	3	9	176	Nut	9

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.

nicht in 3. NF

Customers

CustomerNo	FirstName	LastName	CityId	City
1	Brian	Thompson	NYC	New York City
2	Sally	Henderson	NYC	New York City

- Durch Normalformbildung können *unerwünschte* funktionale Abhängigkeiten eliminiert werden.

XML-Modellierung nicht in 1. NF

```
<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>
  <CityId>NYC</CityId>
  <Type>Sales Person</Type>
</EmployeeTuple>
```

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

✓

Relationales Modell in 3. NF

```
<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>
   <CityId>NYC</CityId>
   <Type>Sales Person</Type>
</EmployeeTuple>
```

✓

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

✓

Normalformbildung für XML

- XML-Daten meist nicht in 1. NF
- Normalformbildung aus dem relationalem Modell *nicht* auf XML übertragbar
- Für XML bisher *kein* rigoroses Verfahren der Normalformbildung
- wird XML als Speicherformat verwendet, müssen dennoch Lösch- und Änderungsanomalien vermieden werden
- informelles Modellierungsverfahren für XML: Asset-Oriented Modeling (Daum & Merten, System Architecture with XML, 2003).

Fazit

- unterschiedliche Anforderungen für Austausch- und Speicherformate:
- <u>Austauschformate</u>: sollten kompakt und einfach zu parsen sein
- <u>Speicherformate</u>: sollten Lösch- und Änderungsanomalien vermeiden und wichtigsten Anfragen optimieren
- Austausch- und Speicherformat deshalb häufig sehr unterschiedlich, auch wenn XML verwendet wird.

Fazit

- 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.