



XML und Datenbanken



Heutige Vorlesung

- XML und Datenbanken weites Feld
- Heutige Vorlesung konzentriert sich auf die Frage, was XML von relationalen Datenbanken lernen kann:
- Was sollte beachtet werden, wenn XML zur Datenspeicherung verwendet wird?

vollständige Darstellung des Themas:

- www.rpbourret.com/xml/XMLAndDatabases.htm
- www.rpbourret.com/xml/XMLDatabaseProds.htm

Übersicht

- relationales Datenmodell
- Abbildung relationales Datenmodell → XML
- Datenmodellierung mit XML
- Normalformen als Hilfsmittel zur Datenmodellierung



Relationales Datenmodell

Relationales Datenmodell

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

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



Primär- und Fremdschlüssel

- mindestens eine Spalte einer Tabelle als **Primärschlüssel** ausgezeichnet
- bei mehreren Spalten: **zusammengesetzter Primärschlüssel**
- **Fremdschlüssel**: referenziert Primärschlüssel einer anderen Tabelle

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

Orders		
OrderNo	CustomerNo	ItemNo
121	1	FX100
5	1	FX200

Eindeutigkeit und Existenz

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

Orders		
OrderNo	CustomerNo	ItemNo
121	1	FX100
5	1	FX200

- Primärschlüssel (d.h. deren Werte) müssen existieren
- Primärschlüssel müssen innerhalb der Tabelle eindeutig sein.
- Für jeden Fremdschlüssel muss ein zugehöriger Primärschlüssel existieren.

Typen von Beziehungen

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

- Tabellen (Relationen) repräsentieren Beziehungen zwischen primitiven Daten.
- 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 zwischen Tabellen ausgedrückt?

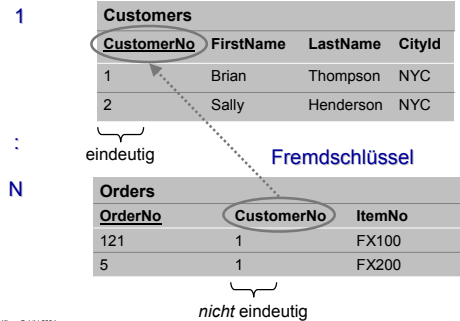
1:N-Beziehung



- 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



1:1-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



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

N:M-Beziehung im relationalen Modell

Customers

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

N:M-Relationship

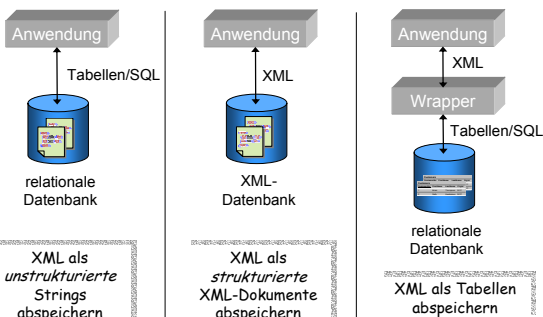
Key	CustomerNo	EmployeeNo
121	1	4
5	1	5

Employees

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

Abbildung relationales Datenmodell → XML

XML & Datenbanken: Drei Alternativen



Wrapper

- Abbildung relationales Datenmodell → XML nötig
- Relationales Datenmodell kann einfach in XML kodiert werden.
- Kodierung könnte auch als Standard für relationale Datenbanken dienen.
- Hierfür gibt es allerdings *keinen* etablierten Standard.
- inoffizieller Vorschlag:
<http://www.w3.org/XML/RDB.html>

Abbildung Relationales Modell → 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>
</Database>

```

- 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

© Klaus Schild 2004

19

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>
</Database>

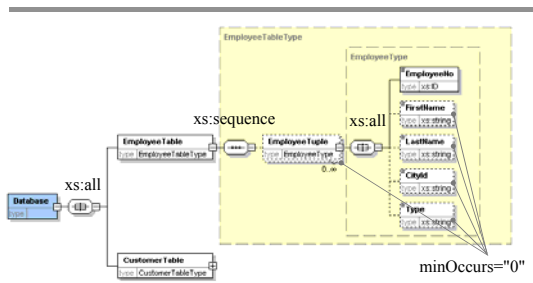
```

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

© Klaus Schild 2004

20

Das zugehörige XML-Schema



- Reihenfolge der Tabellen, Tupel und Spalten egal

© Klaus Schild 2004

21

Kodierung von Schlüssel

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

Probleme

- keine *zusammengesetzten* Primärschlüssel darstellbar
- Statt Eindeutigkeit innerhalb der Tabelle, erzwingt xs:ID Eindeutigkeit in gesamter Datenbank
- Zwei Tabellen dürfen also niemals identischen Primärschlüssel haben.
- Statt eines ganz bestimmten Primärschlüssels, referenziert xs:IDREF *beliebigen* Primärschlüssel mit Typ xs:ID.

© Klaus Schild 2004

22

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 in *gesamter* Datenbank eindeutig sein.

Fremdschlüssel kann sich auf einen *beliebigen* Primärschlüssel beziehen.

© Klaus Schild 2004

23

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 Employee-Tabelle 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.

© Klaus Schild 2004

24

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**: referenzierter Primärschlüssel
- **xs:selector**: spezifiziert Kontext, auf die die Existenzbedingung angewandt wird.
- ein oder mehrere **xs:field**-Elemente: Elemente/Attribute, für die Primärschlüssel (mit passendem Typ) existieren muss

Fazit

- Abbildung relationale Datenbank → XML ohne Informationsverlust möglich (einschl. Primär- und Fremdschlüssel)
- Instanz kodiert Datenbank
- XML-Schema kodiert Datenbankschema

Datenmodellierung mit XML

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 *nicht* erlaubt.
- XML erlaubt solche geschachtelten Strukturen.

Warum nicht die hierarchische Strukturierungsmöglichkeiten von XML voll ausnutzen?

Beispiel

```
<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>
```

- Könnte 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 er vermittelt hat.
- Gefahr, ungewollt Informationen zu löschen
- Daher Order-Relation auslagern und hier 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.
- unnötiger Verwaltungsaufwand
- Gefahr von Inkonsistenzen
- Daher City-Relation auslagern und hier durch Fremdschlüssel ersetzen.

Verbesserte Modellierung

```

<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.
- Deshalb Zuordnung ItemNo/ItemName auslagern und durch Fremdschlüssel 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>
  
```

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

Wer ist Vermittler?

- Angestellten-Datenbank muss 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>
  <CityKey>C123</CityKey>
  <Type>Sales Person</Type>
</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>

<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 Modell

```

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

<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>
  
```

Relationale Modellierung

```

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

<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>
  
```

N:M

N:M-Beziehung 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>
  <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

Ausgangspunkt

- strukturiertes XML-Dokument als Speicherformat

Transformationen

- zur Beseitigung von Lösch- und Änderungsanomalien

Ergebnis

- mehrere, wesentlich flachere XML-Strukturen
- relationalem Datenmodell ähnlich
- Ausnahme: N:M-Beziehungen als geschachtelte Strukturen

Normalformen

Normalformen

- Hilfsmittel zur Datenmodellierung
- für relationales Modell formal definiert

Ziele

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

Mittel

- Verständnis der Bedeutung der Daten
 - Verständnis funktionaler Abhängigkeiten zwischen Feldern
- Auf XML übertragbar?

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

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 Daten selbst zu erkennen, sondern daran, wie sie verwendet werden.

Weitere funktional Abhängigkeiten

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 funktional abhängig
- alle andere Felder nur von Teilen des Primärschlüssels funktional abhängig

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.

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

nicht in 2. NF

© Klaus Schild 2004

49

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.

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

nicht in 3. NF

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

© Klaus Schild 2004

50

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>

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

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

© Klaus Schild 2004

51

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>
  <CityKey>C123</CityKey>
  <Type>Sales Person</Type>
</EmployeeTuple>

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

© Klaus Schild 2004

52

Normalformbildung für XML

- XML-Daten normalerweise *nicht* in 1. NF
- Normalformbildung aus dem relationalem Modell *nicht* auf XML übertragbar
- Für XML bisher *kein* rigoroses Verfahren der Normalformbildung
- informelles Modellierungsverfahren für XML: Asset-Oriented Modeling (Daum & Merten, System Architecture with XML, 2003).

© Klaus Schild 2004

53

Fazit

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

© Klaus Schild 2004

54

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.

Wie geht es weiter?



heutige Vorlesung

- ☑ Was sollte beachtet werden, wenn XML zur Datenspeicherung verwendet wird?

heute 16:15

- betreute Rechnerübung

nächste Woche

- Web Services (insgesamt 4 Termine)