
XML und Datenbanken

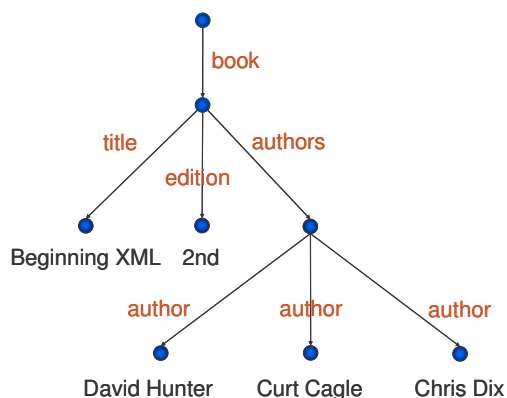
Übersicht

-
- Wie XML persistent speichern?
 - Vergleich XML mit relationalem Modell
 - Exkurs: relationales Modell
 - Darstellung von N:M-Beziehungen
 - funktionale Abhängigkeiten, Normalformen
 - Wie Daten mit XML modellieren?
 - Schwerpunkt: Vergleich XML mit relationalem Modell
 - vollständige Darstellung des Themas:
www.rpbouret.com/xml/XMLAndDatabases.htm

Wie XML persistent speichern?

XML vs. relationale Datenbanken

XML



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

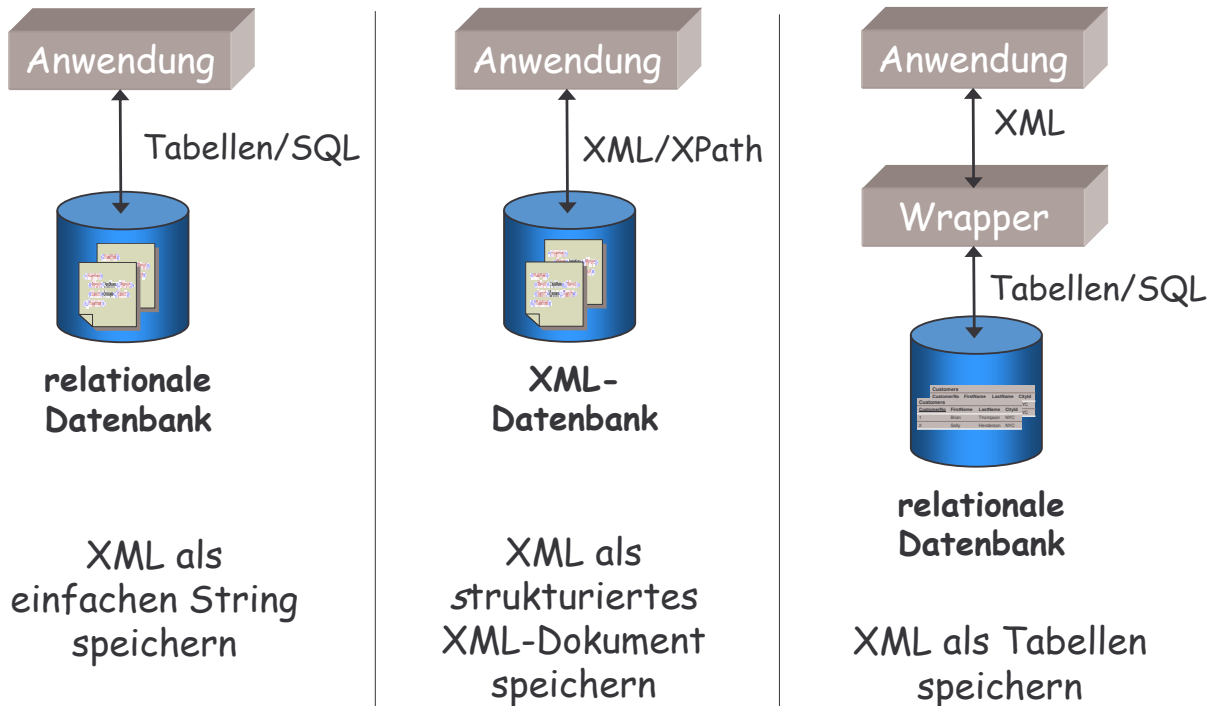
relationale Datenbanken

| 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 |

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

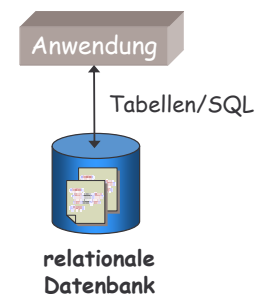


1. XML als String speichern

- hierarchische XML-Struktur als Wert eines Feldes in einer Tabelle speichern
- XML-Struktur als String serialisieren

Vor- und Nachteile

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

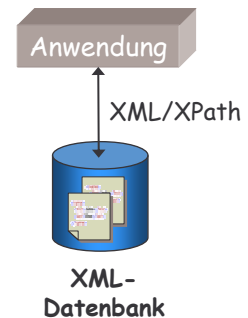


2. XML in XML-Datenbank speichern

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

Vor- und Nachteile

- + komplexe Anfragen auf XML-Struktur möglich
- + XPath wird unterstützt
- neue Datenbank nötig
- XML-Datenbanken nicht interoperabel
- schwierige Integration mit bestehenden relationalen Datenbanken (RDB)
- keine Systematik der Datenmodellierung

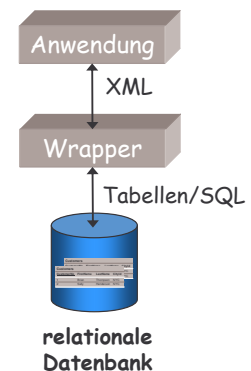


3. XML als Tabellen speichern

- Abbildung XML → Tabellen möglich
- Abbildung Tabellen → XML problemlos
- Anfragen: SQL mit XML-Funktionen (z.B. SQL/XML)

Vor- und Nachteile

- + vorhandene RDB kann genutzt werden
- + von modernen RDBMS unterstützt
- + Systematik der Datenmodellierung für Tabellen
- Abbildung XML → Tabellen → XML liefert nicht unbedingt ursprüngliche XML-Struktur



- Frage ist, wie tief XML-Strukturen in die Datenbank integrieren?
 1. XML als einfachen String in Feld einer Tabellen speichern: **gar nicht integrieren**
 2. XML-Datenbanken: **voll integrieren**
 3. XML als Tabellen speichern: **soweit wie möglich integrieren**
- nur zu beantworten, wenn XML mit relationalem Datenmodell verglichen wird

Exkurs: Relationales Modell

| 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 einer bestimmten Zeile
- bei mehreren Spalten: **zusammengesetzter Primärschlüssel**
- **Fremdschlüssel**: referenziert Primärschlüssel anderer 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 jede Zeile 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 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 im relationalem Modell ausgedrückt?



- 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.

Primärschlüssel

1

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

:

eindeutig

Fremdschlüssel

N

| Orders | | |
|----------------|------------|--------|
| <u>OrderNo</u> | CustomerNo | 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 besteht eine 1:1-Beziehung.

1:1-Beziehung im relationalen Modell

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

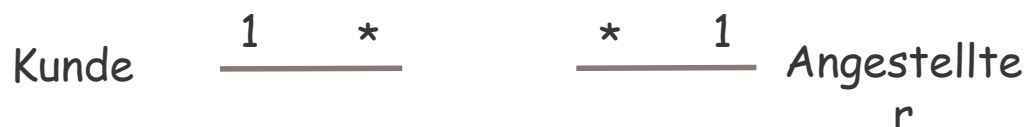
| Customers (detailed) | | | | | |
|----------------------|----------|-----------|--------|------------|--------------|
| <u>CustomerNo</u> | LastName | FirstName | CityId | CreditCard | CreditCardNo |
| 1 | Brian | Thompson | NYC | Amex | 10090040234 |
| 2 | Sally | Henderson | NYC | Visa | 100800402e3 |

3

- beide Schlüssel gleichzeitig Primär- und Fremdschlüssel

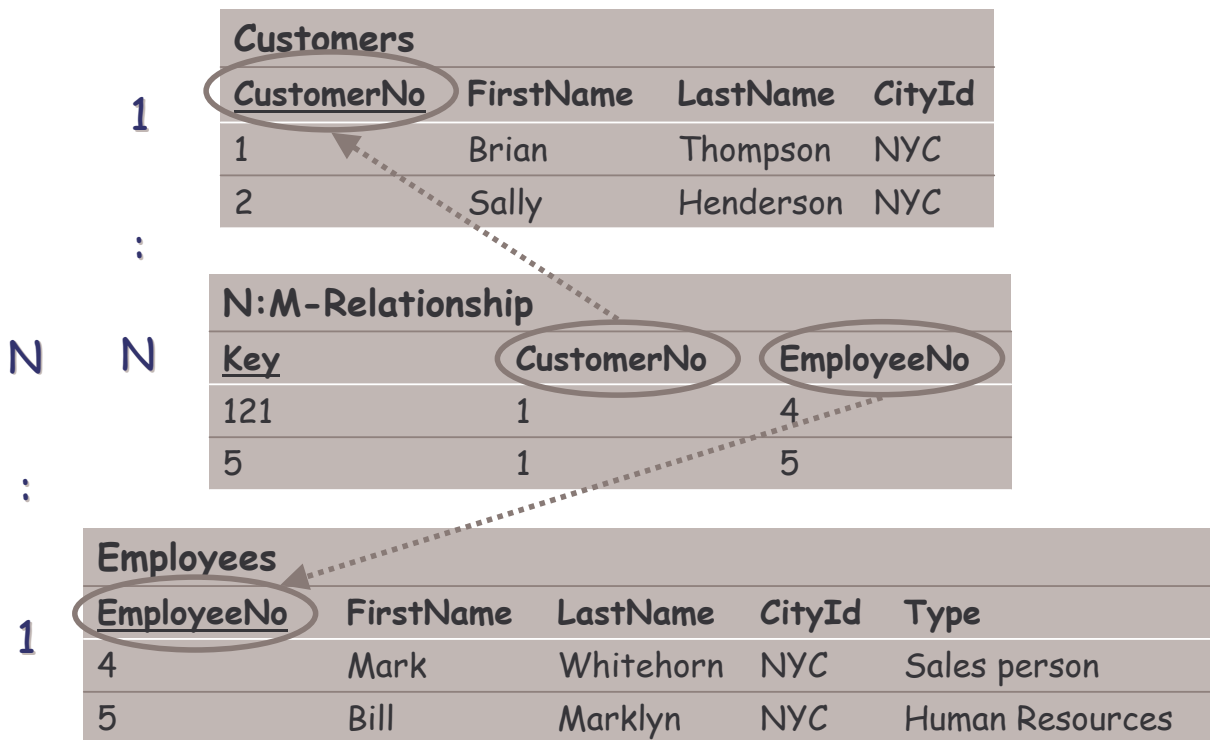


- Angestellter kann mehrere Kunden betreuen.
 - Umgekehrt kann Kunde gleichzeitig von mehreren Angestellten betreut werden.
- ⇒ Zwischen Kunden und Angestellten besteht eine N:M-Beziehung.

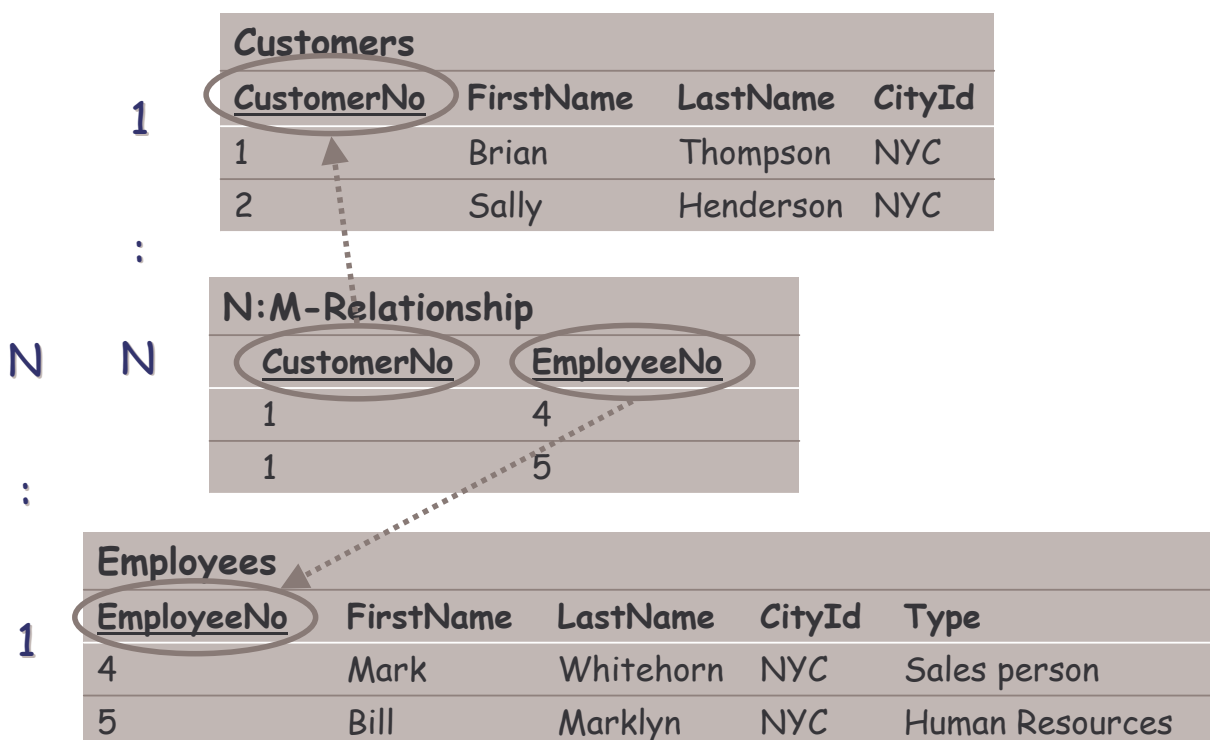


- im relationalen Modell N:M-Beziehung nicht direkt darstellbar
- muss in zwei 1:N-Beziehungen aufgebrochen werden
- Dritte Tabelle enthält Fremdschlüssel beider Tabellen.

N:M-Beziehung im relationalen Modell



Alternative Darstellung



- Hilfsmittel zur Modellierung von Daten
- für relationales Modell formal definiert

Ziel

- 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

Funktionale Abhängigkeiten

- Beispiel: Fläche = Höhe \times 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 Tabelle

| 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
- ⇒ $EmployeeNo = f(OrderNo)$
- ⇒ $EmployeeNo$ funktional abhängig von $OrderNo$.

Wichtig: Funktionale Abhängigkeiten nicht an Daten selbst zu erkennen, sondern an ihrer Verwendung.

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 |

- $Quantity = f_1(OrderNo, ItemNo)$
- $ItemName = f_2(ItemNo)$
- $CustomerNo = f_3(OrderNo)$

- 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.

nicht erlaubt:

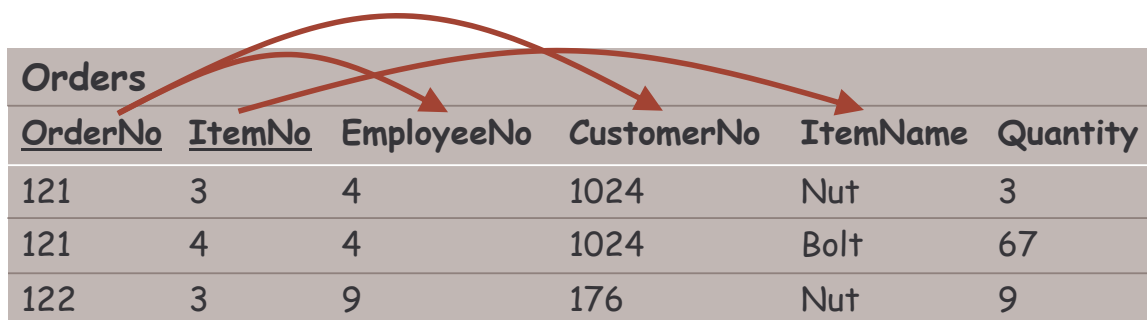
notwendig:

- $F_i = f(P)$

- $F_i = f(P')$, P' echte Teilmenge von P
- $F_i = f(P)$, jedoch $F_i = f(F_j)$, $F_j = f(P)$ 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 Modell → XML

- Relationales Modell 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>
  ...
```

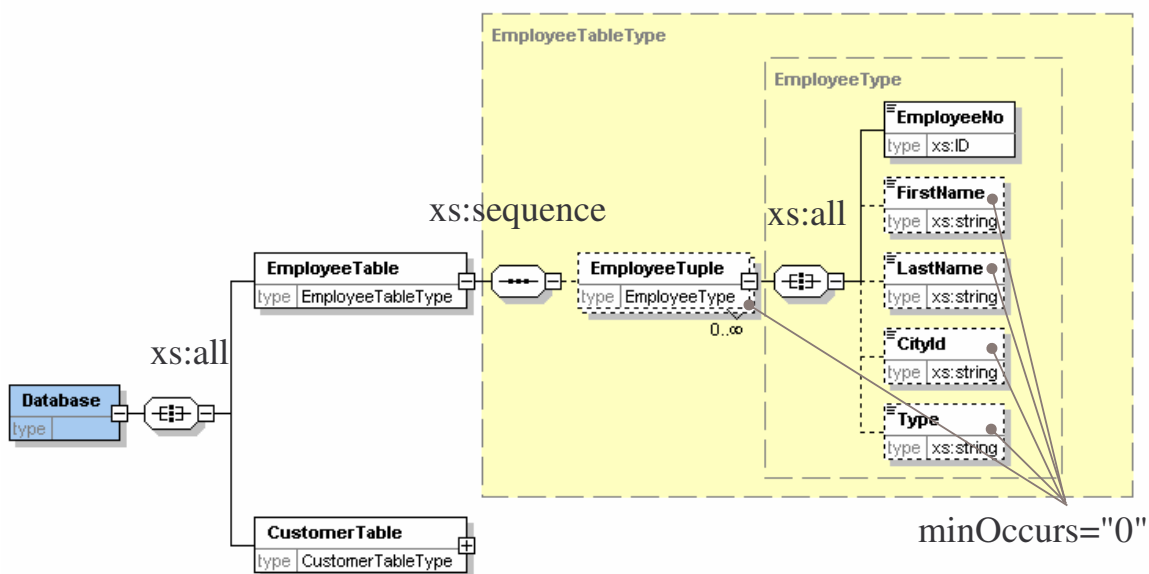
- Wurzel-Element = Name der Datenbank
- für jede Tabelle genau ein Kind-Element
- darunter für jedes Tupel genau ein Kind-Element
- darunter für jede Spalte ein Kind-Element

```

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

Das zugehörige XML-Schema



- Reihenfolge der Tabellen, Tupel und Spalten egal

- Primärschlüssel: Typ `xs:ID`.
- Fremdschlüssel: 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 beziehen.

```
<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 sind

```
<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>
```

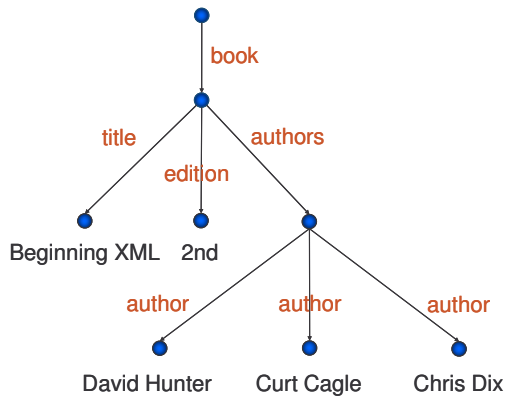
⇒ Wert von CityId immer definiert

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

- einfach und ohne Informationsverlust möglich
- gilt auch für Primär- und Fremdschlüssel
- XML-Dokument (Instanz) kodiert Datenbank
- XML-Schema kodiert Datenbankschema

Datenmodellierung mit XML

XML



relationales Modell

| 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 Modell:

Relationales Modell

- 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 die hierarchischen Möglichkeiten von XML nicht 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>
```


- Könnte so 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 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.

```
<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 hier durch Fremdschlüssel ItemNo ersetzen.

```
<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>
```



```
<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 normalerweise Ansprechpartner für Kundenauftrag

Wer ist Vermittler?

⇒ Angestellten-Datenbank muss durchsucht werden, um Vermittler 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>

<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>

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

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

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

Vergleich mit relationalem Modell

```
<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 Modell (in 3. NF) sehr ähnlich
- Ausnahme: N:M-Beziehungen als geschachtelte Strukturen

relationales Modell

- schrittweise Normalformbildung: Ergebnis formal definiert

XML

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

Fazit aus heutiger Vorlesung

Daten mit vielen funktionalen Abhängigkeiten

- bewährte Speicherformate wie relationale Datenbanken besser
- Tabellen als XML serialisieren

Text-Dokumente mit nur wenigen funktionalen Abhängigkeiten

- XML auch als Speicherformat geeignet

heutige Vorlesung

- ☑ Wie XML persistent speichern?
- ☑ Vergleich XML mit relationalem Modell
- ☑ Wie Daten mit XML modellieren?

heutige Übung (1. Gruppe)

- XSLT

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.

nicht in 2. NF

| <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 |

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