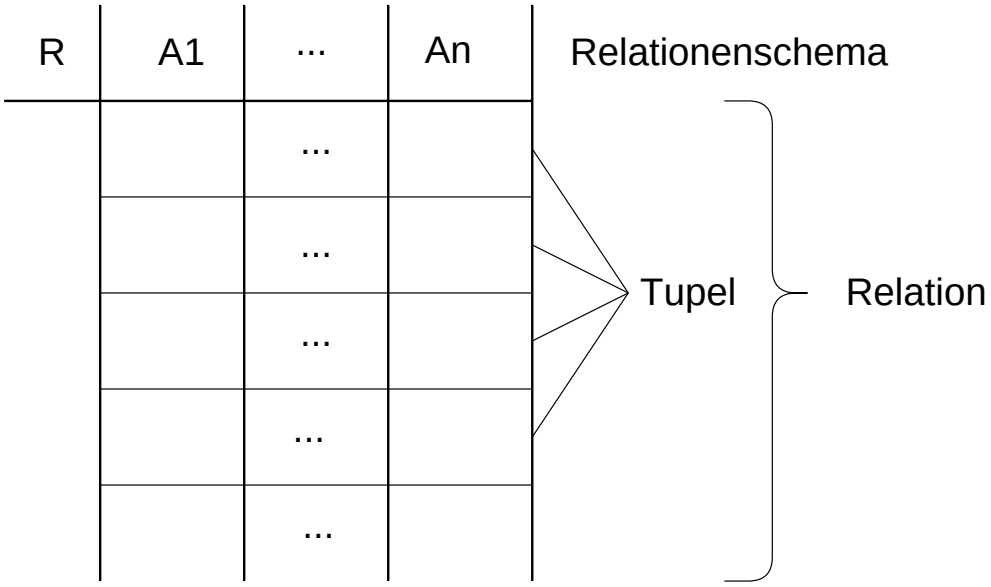
2.2 Relationen-Modell

Grundlage für viele DBMS wie z.B. DB2, INGRES, ORACLE, INFORMIX, MySQL, ...

Konzepte:

Relationen	Mengen von Entities
Tupel	Entities
Attribute	Werte

Erlaubt: Verknüpfung von beliebigen Tupeln über Wertevergleiche



Übersetzung von ER-Schemata in Relationale Schemata

0. Schlüselattribute festlegen

1. Entitytyp $E(A_1, ..., A_k) \Rightarrow$

   Relationenschema $E(A_1, ..., A_k)$

   vorhandene Schlüssel übernehmen

2. Beziehungstyp $R(E_1, ..., E_l, A_1, ..., A_m) \Rightarrow$

   Relationenschema $R(X_1, ..., X_l, A_1, ..., A_m)$

   wobei X_i Schlüssel(attributemenge) von E_i $(i = 1..l)$

3. gegebenenfalls Relationen streichen und/oder zusammenfassen; Attribute bzw. Relationen umbenennen

spezielle Behandlung von funktionalen Beziehungen und Generalisierungen

2.3 Netzwerk-Modell

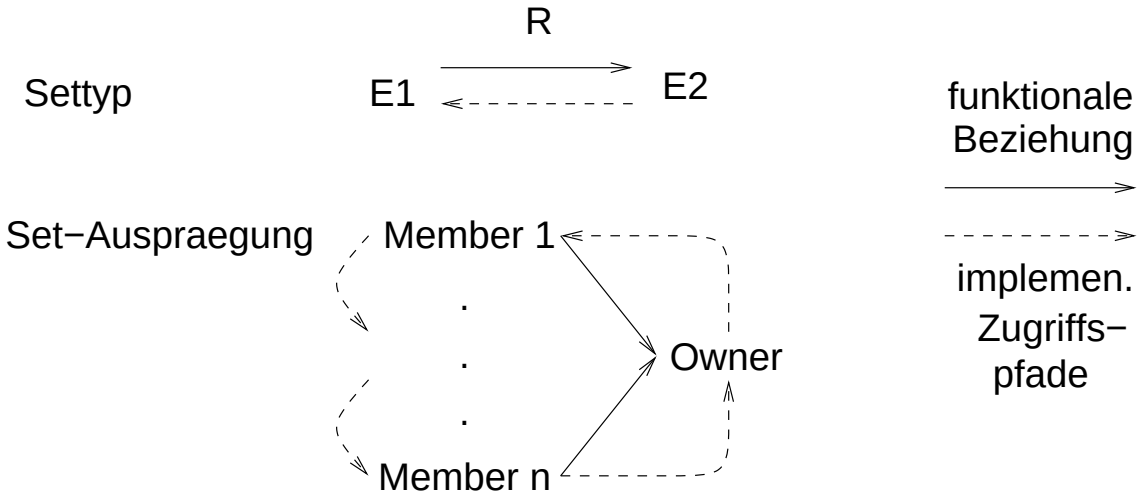
immer noch Grundlage von vielen heute eingesetzten Systemen mit DBMS-Unterstützung

Konzepte: im Vergleich zum ER-Modell nur Attribute, Entitytypen und zweistellige funktionale Beziehungen

Terminologie:

Feld	$\approx$	Attribut
logischer Satz	$\approx$	Entity
logischer Satztyp	$\approx$	Entitytyp
logisches Satzschema	$\approx$	Entitytyp mit Attributen $E(A_1, ..., A_n)$
Settyp	$\approx$	2-stellige fkt. Beziehung $R : E_1 \to E_2$

Darstellung von Set-Ausprägungen: Gruppen von sogenannten Owner/Member-Sätzen mit logischen Zeigern



Verknüpfung von Sätzen nur über logische Zeigerketten

Übersetzung von ER-Schemata in Netzwerk-Schemata

1) Entitytyp $E(A_1, ..., A_k)$

   $\Rightarrow$

   Satztyp $E(A_1, ..., A_k)$

2.a) funktionaler Beziehungstyp $R : E_1 \to E_2$

   $\Rightarrow$

   Settyp $E_1 \to E_2$

2.b) anderer Beziehungstyp $R(E_1, ..., E_l, A_1, ..., A_m)$

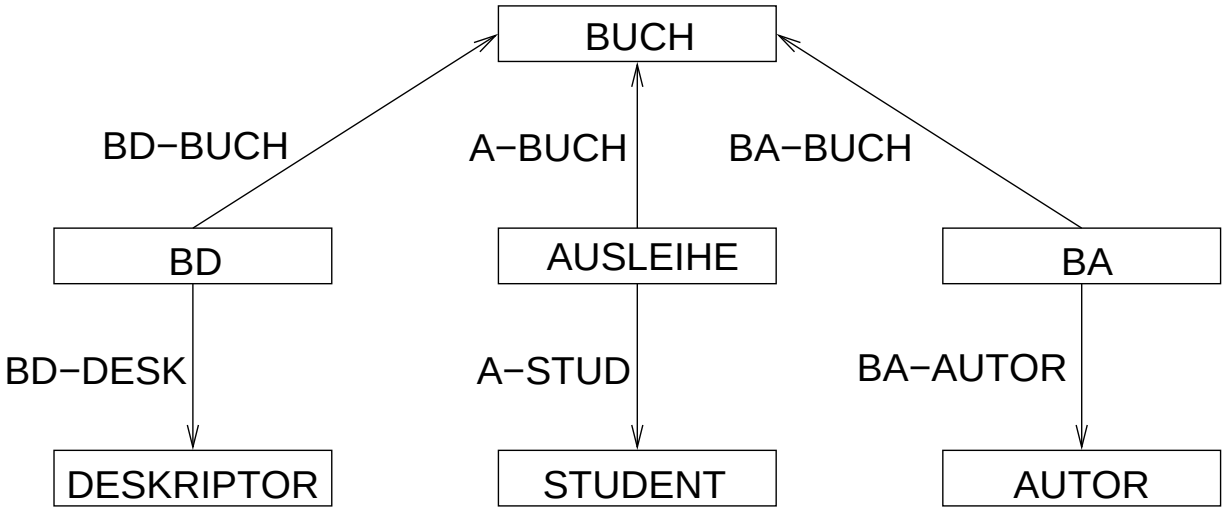
   $\Rightarrow$

   neuer Satztyp $E_0(A_0, A_1, ..., A_m)$ (Kett-Entity)

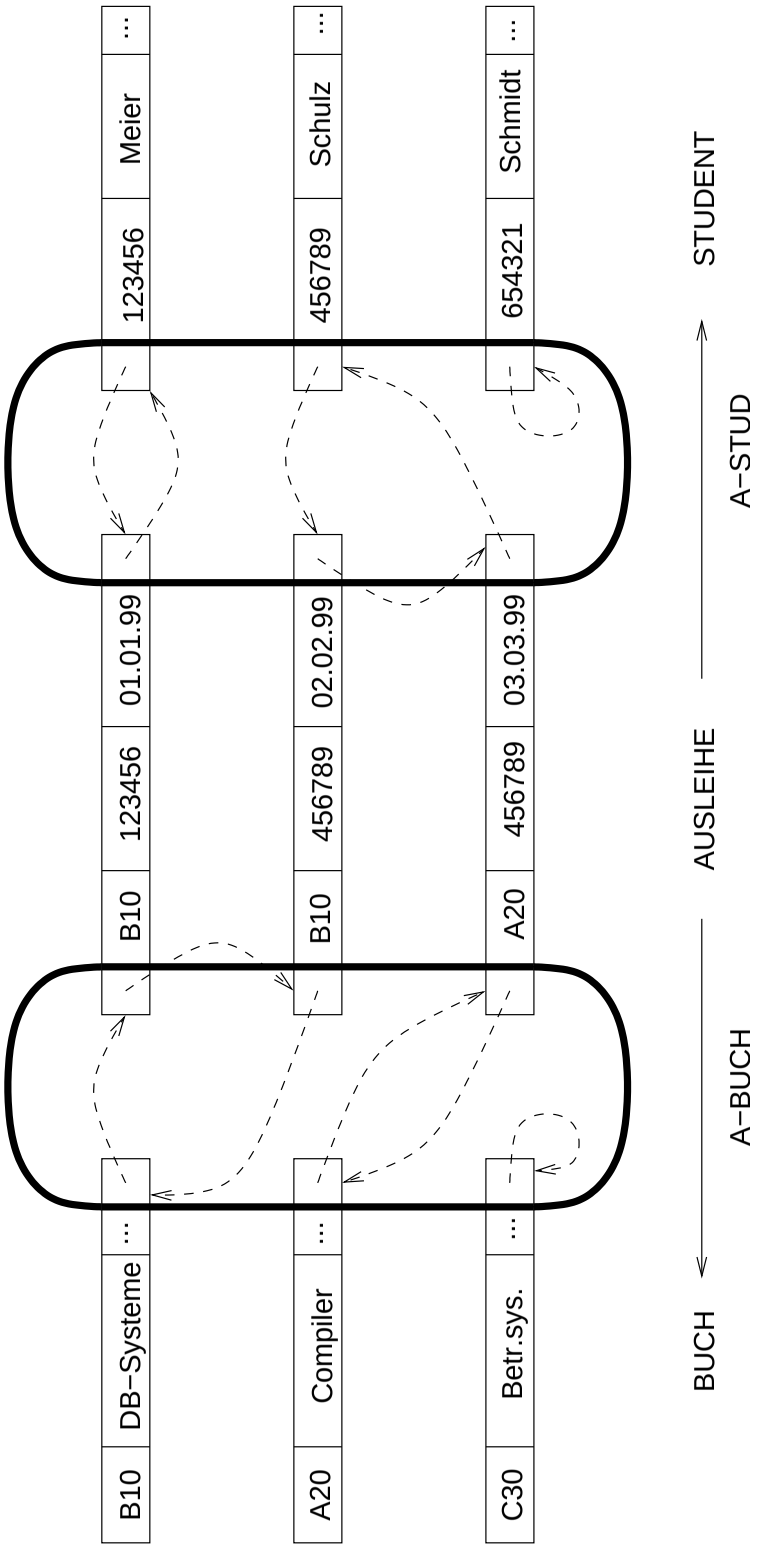
   A_0 Feld(er) mit eindeutiger Kennzeichnung (z.B. Schlüssel von $E_1, ..., E_l$)

   Settypen $S_i : E_0 \to E_i$ $(i = 1..l)$

Netzwerk-Schema der Bibliotheks-DB



Beispielzustand (Ausschnitt) im Netzwerk-Modell



2.4 Hierarchisches Modell

Grundlage für IMS von IBM; immer noch weit verbreitet

Konzepte:

- im Vergleich zum Netzwerkmodell: nur Hierarchien (Wälder) von Settypen zulässig
- Abstraktion von Dateien mit Wiederholungsgruppen zu Dateihierarchien

Problem: Zugriffe nur gemäß Hierarchie möglich

Auswege:

– Duplizierung von Sätzen

\+ Verwendung von virtuellen Satztypen; entsprechen Zeigertypen zu Sätzen des entspr. Satztyps

Übersetzung von Netzwerk-Schemata in Hierarchische Schemata

1. Zerlegung des Netzwerks in Bäume
2. gegebenenfalls Duplizierung von Knoten mit `virtual`
3. gegebenenfalls Verbesserungen

2.5 Objektorientierte Modelle

Sichtweise: Speziallfall des ER-Modells in dem statt allgemeiner Relationships nur Funktionen (allerdings auch mengenwertige) zugelassen sind

beispielsweise statt

```
gebucht-fuer(PASSAGIER,ABFLUG)
```

nun

```
PASSAGIER attribute Abfluege:set(ABFLUG)

ABFLUG attribute Passagiere:set(PASSAGIER)
```

in einige Ansätzen auch **inverse** Funktionen möglich

```
PASSAGIER attribute Abfluege:set(ABFLUG)
          inverse ABFLUG::Passagiere

ABFLUG attribute Passagiere:set(PASSAGIER)
       inverse PASSAGIER::Abfluege
```

Allgemeine Konzepte objekt-orientierter DBMSe

- **Objektidentitäten**; im Lebenslauf von Objekten nicht veränderbar; siehe Platzhalter für Objekte im ER-Modell ($\mu(E)$)

- **Objektstrukur und Konstruktoren**

```
tuple(t1,..,tn), set(t), list(t), ...
```

```
define type Employee :
  tuple( name:string,
         ssn :string,
         birthdate:Date,
         sex:char,
         dept:Department )
define type Date :
  tuple( year:integer,
         month:integer,
         day:integer )
define type Department :
  tuple( dname:string,
         dnumber:integer,
         mgr:tuple( manager:Employee,
                    startdate:Date ),
         locations:set(string),
         employees:set(Employee),
         projects:set(Project) )
```

- **Typen versus Klassen**

  Typen legen generelle Struktur von Objekten fest

  Klassen beziehen sich auf Typen, sind zusätzlich (in jedem DB-Zustand) mit einer endlichen Mengen von entsprechenden Objekten bevölkert

```
define class YoungEmployee
  type Employee
  ...
define class CompSciEmployee
  type Employee
  ...
```

- **Einkapselung und Methoden**

  Dienste eines Objektes durch Signatur festgelegt; Verbergen der Implementierung; Objekt damit Einheit von **Struktur und Verhalten**

Bespiel: Employee, Department

```
define type Employee

  tuple( name:string,
         ssn:string,
         birthdate:Date,
         sex:char,
         dept:Department ),

  operations
    age():integer,
    createEmployee():Employee,
    destroyEmployee():boolean,
    ...

define type Department

  tuple( dname:string,
         dnumber:integer,
         mgr:tuple(manager:Employee,
                   startdate:Date ),
         locations:set(string),
         employees:set(Employee),
         projects:set(Project) )

  operations
    numberOfEmployees():integer,
    createDepartment():Department,
    destroyDepartment():boolean,
    addEmployee(e:Employee):boolean,
    removeEmployee(e:Employee):boolean,
    ...
```

- **Typ- und Klassen-Hierarchien**

```
define type Employee subtypeOf Person
  ...
define type Student subtypeOf Person
  ...
define class CompSciEmployee subclassOf UniEmployee
  ...
```

- **Polymorphismus**

```
define type Line
  ...
  operations display() ...
  ...
define type Rectangle
  ...
  operations display() ...
  ...
```

- **Mehrfach-Vererbung**

```
define type EmployedStudent
  subtypeOf Employee, Student
```

- **Versionen und Konfigurationen**

  verschiedene Versionen eines komplexen Objektes z. B. für unterschiedliche Zeitpunkte

  verschiedene Konfigurationen eines komplexen Objektes z. B. ein Software-Objekt angepaßt an eine spezielle Hardware

2.6 Semistrukturierte Datenmodelle

Merkmale von semistrukturierten Datenmodellen

- Schema der Daten muß nicht zentral gespeichert sein, sondern kann auch in jedem Dokument vorhanden sein

- Daten können wechselnde Struktur haben

- Daten können auch nur sehr wenig Struktur haben, wie z.B. der Volltext eines Zeitschriftenartikels

- Anzahl der möglichen Attribute und Vielfalt der internen Struktur kann sehr groß sein

- Attribute und Strukturierung von Daten unterliegen oft häufigen Änderungen

- Unterschied zwischen Daten und Schema wird häufig unscharf, er ist zumindest nicht so klar wie bei klassischen Datenmodellen

Semistrukturierte Daten am Beispiel XML

Strukturierungsmittel:

- Sequenz (A1,A2)
- Alternative (A1|A2)
- Wiederholung
  - beliebige Iteration A*
  - nichtleere Iteration A+
  - optionales Element A?

entstehende Ausdrücke sind ähnlich zu regulären Ausdrücken

Beispiel DTD (Document Type Definition)

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE Buch [
<!-- Buch-DTD -->
<!ELEMENT Buch (ISBN, Titel, Verlag, Autor+,
  Stichwort*, Version*, Abstract, Buchtext?)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Titel (Haupttitel, Untertitel?)>
<!ELEMENT Haupttitel (#PCDATA)>
<!ELEMENT Untertitel (#PCDATA)>
<!ELEMENT Verlag (#PCDATA)>
<!ELEMENT Autor (#PCDATA)>
<!ELEMENT Stichwort (#PCDATA)>
<!ELEMENT Version (Auflage, Jahr, Seiten, Preis)>
<!ELEMENT Auflage (#PCDATA)>
<!ELEMENT Jahr (#PCDATA)>
<!ELEMENT Seiten (#PCDATA)>
<!ELEMENT Preis (#PCDATA)>
<!ELEMENT Abstract (#PCDATA)>
<!ELEMENT Buchtext (Kapitel*)>
<!ELEMENT Kapitel (Ueberschrift, Abschnitt*)>
<!ELEMENT Ueberschrift (#PCDATA)>
<!ELEMENT Abschnitt (#PCDATA)>
]>
```

Beispiel eines selbstbeschreibenden Datensatzes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE bib [
<!ELEMENT bib (book+)>
<!ELEMENT book (author+, title, year, publisher)>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT author (firstname?, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (name, address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
]>

<bib>
  <book isbn="3-929821-31-1">
    <author> <firstname>Andreas</firstname>
             <lastname>Heuer</lastname>
    </author>
    <author> <lastname>Saake</lastname>
    </author>
    <title>Datenbanken: Konzepte und Sprachen</title>
    <year>2000</year>
    <publisher>
      <name>International Thomson Publishing</name>
      <address>Bonn</address>
    </publisher>
  </book>
</bib>
```

Details

- Muster einer Attributdefinition

  ```
  <!ATTLIST book isbn CDATA #REQUIRED>
  ```

  Dokumenteinheit

  Attributename

  Attributdatentyp

  Angabe zu Optionalität

- Angabe eines Attributwertes

  ```
  <book isbn="3-929821-31-1">
  ```

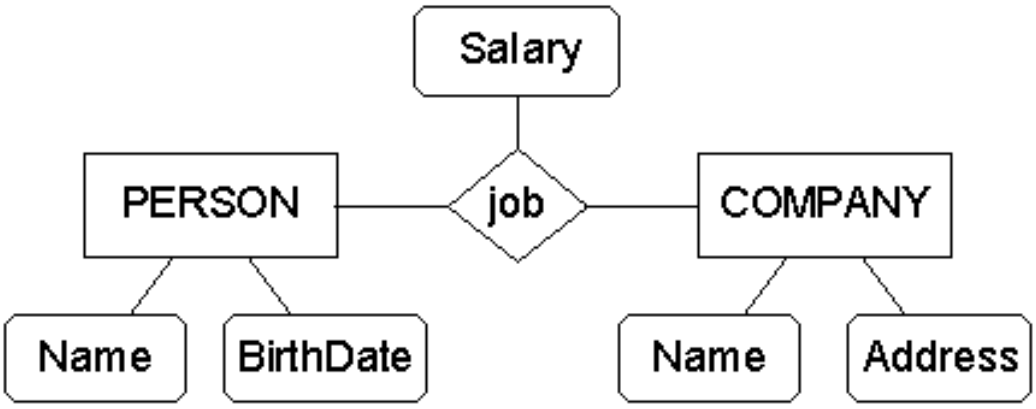
- Tags (Attribute) definieren durch Klammerstruktur hierarchisch strukturierte Dokumente

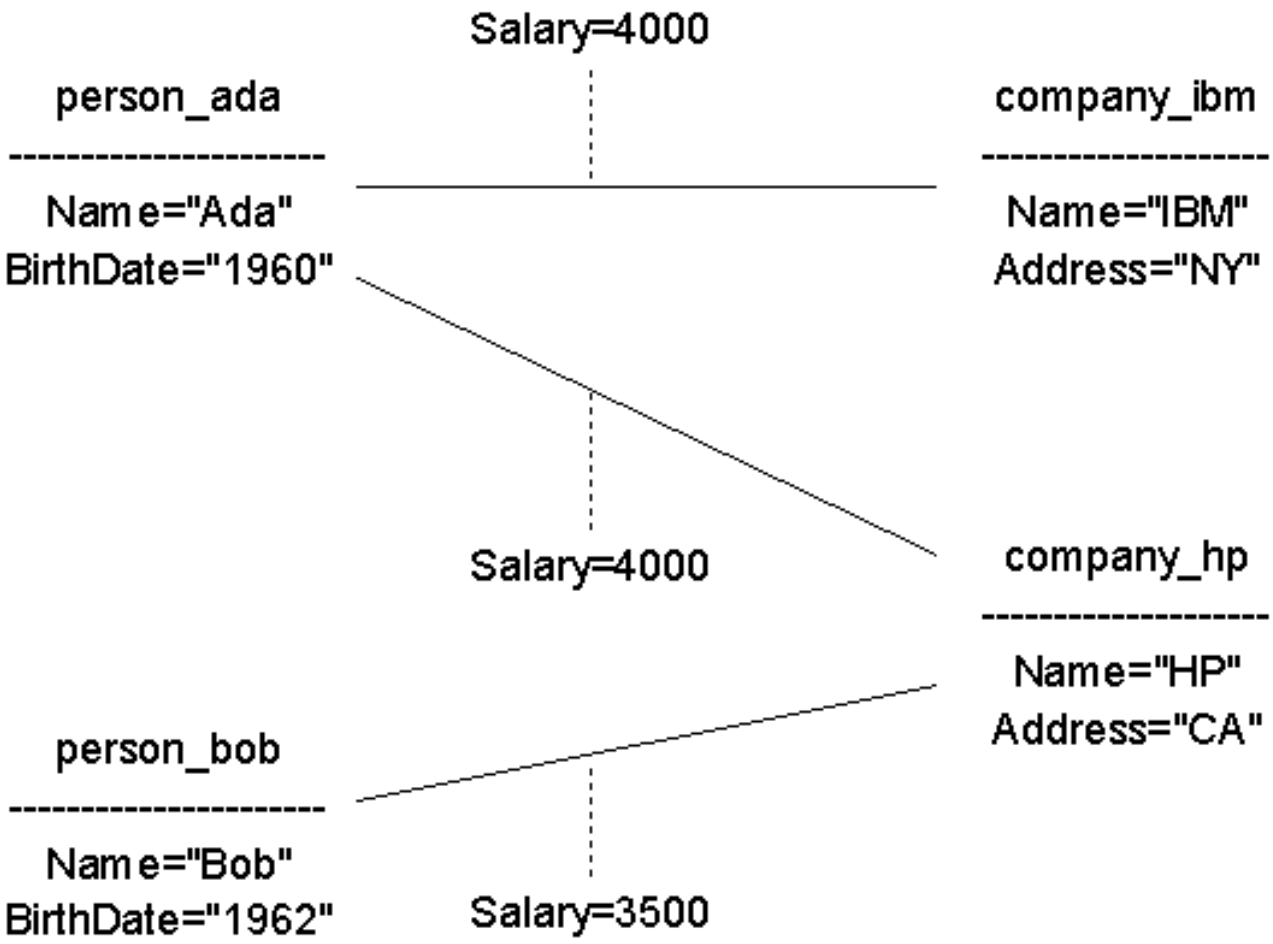
  ```
  <author>
    <firstname>Andreas</firstname>
    <lastname>Heuer</lastname>
  </author>;
  ```

- DTDs stellen einen Ansatz dar, mit XML Daten zu beschreiben

- weiterer Ansatz ist W3C-Standard XML Schema

Objekte in XML - ER-Schema und ER-Zustand





Objekte in XML - XML-DTD

```
<?xml version="1.0" encoding="iso-8859-1" ?>

<!DOCTYPE JobWorld[

<!ELEMENT JobWorld ((COMPANY | PERSON | job)*)>

<!ELEMENT PERSON (#PCDATA)>
<!ATTLIST PERSON PERSON.id ID    #REQUIRED
                 Name      CDATA #REQUIRED
                 BirthDate CDATA #REQUIRED>

<!ELEMENT COMPANY (#PCDATA)>
<!ATTLIST COMPANY COMPANY.id ID    #REQUIRED
                  Name       CDATA #REQUIRED
                  Address    CDATA #REQUIRED>

<!ELEMENT job (#PCDATA)>
<!ATTLIST job PERSON.refid  IDREF #REQUIRED
              COMPANY.refid IDREF #REQUIRED
              Salary        CDATA #REQUIRED>

]>
```

Objekte in XML - XML-Dokument

```
<JobWorld>
<PERSON PERSON.id="person_ada"
        Name="Ada"
        BirthDate="1960"/>
<PERSON PERSON.id="person_bob"
        Name="Bob"
        BirthDate="1962"/>
<job PERSON.refid="person_ada"
     COMPANY.refid="company_ibm"
     Salary="4000"/>
<job PERSON.refid="person_ada"
     COMPANY.refid="company_hp"
     Salary="4000"/>
<job PERSON.refid="person_bob"
     COMPANY.refid="company_hp"
     Salary="3500"/>
<COMPANY COMPANY.id="company_ibm"
         Name="IBM"
         Address="NY"/>
<COMPANY COMPANY.id="company_hp"
         Name="HP"
         Address="CA"/>
<!-- nice things above, ugly things below -->
<job PERSON.refid="person_bob"
     COMPANY.refid="company_hp"
     Salary="3500"/>
<job PERSON.refid="person_ada"
     COMPANY.refid="person_ada"
     Salary="3500"/>
</JobWorld>
```

2.7 Vergleich der Datenmodelle

Kriterien

1. Einfachheit der Benutzung

2. Sprachebene der DB-Sprachen

3. Effizienz der Implementierung

	Benutzung	Sprachebene	Effizienz
REL	einfach	hoch (A)	sehr gut
NW	komplex	mittel (B)	gut
HIER	umständlich	sehr niedrig (C)	(sehr gut) (D)
OO	(einfach)	hoch	gut

(A) mengenorientiert, kein Zwang zur Verwendung von Zugriffspfaden, beliebige Verknüpfungen

(B) satzorientiert, Navigation über logische Zugriffspfade

(C) implementierungsnah

(D) nur bei vorgesehenen Zugriffspfaden, sonst praktisch unmöglich

Vergleichende erste Beispielanfragen:

Welche Autoren liest der Student Zimmermann?

```
ER-MODELL
select AName(AUTOR(ba))
from   ba in BA
where  exists (aus in AUSLEIHE)
         ( BUCH(ba)=BUCH(aus) and
           SName(STUDENT(aus))="Zimmermann" )

RELATIONENMODELL
select AName
from   AUTOREN, AUSLEIHE, STUDENT
where  AUTOREN.Doknr = AUSLEIHE.Doknr
and    AUSLEIHE.Matnr = STUDENT.Matnr
and    STUDENT.SName = "Zimmermann"

NETZWERKMODELL
SName = "Zimmermann"
find any STUDENT using SName
if found then
  find first AUSLEIHE within A-STUD
  while found do
    find owner BUCH within A-BUCH
    find first BA within BA-BUCH
    while found do
      find owner AUTOR within BA-AUTOR
      print AName
      find next BA within BA-BUCH
    od
    find next AUSLEIHE within A-STUD
  od
fi
```