

Vorlesung Multimedia-Datenbanken

Wiederholung: Relationale Datenbanksysteme und Objekt-relationale Datenbanksysteme

(Folien nach Kemper, Eickler)



Relationenschema:

Telefonbuch:

{[Name: string, Adresse: string, Telefonnr: integer]}

Operationen:

- ◆ Kartesisches Produkt
- ◆ Selektion
- ◆ Projektion
- ◆ Umbenennung
- ◆ Differenz, Vereinigung, Schnittmenge

SQL

Select	Projektion
from	kartesischesProdukt
where	Selektion

Große Objekte (Large OBjects, LOBs)]

- ◆ Hierbei handelt es sich um Datentypen, die es erlauben, auch sehr große Attributwerte für z.B.~Multimedia-Daten zu speichern. Die Größe kann bis zu einigen Giga-Byte betragen. Vielfach werden die Large Objects den objekt-relationalen Konzepten eines relationalen Datenbanksystems hinzugerechnet, obwohl es sich dabei eigentlich um "reine" Werte handelt.

Create Table Professoren

```
( PersNr integer primary key,  
  Name varchar(30) not null,  
  Rang character(2) check  
    (Rang in ('C2', 'C3', 'C4')),  
  Raum integer unique,  
  Passfoto BLOB(2M),  
  Lebenslauf CLOB(75K) );
```

```
LOB (Lebenslauf) store as  
( tablespace Lebensläufe  
  storage (initial 50M next 50M) );
```

CLOB

- ◆ In einem Character Large Object werden lange Texte gespeichert.
- ◆ Der Vorteil gegenüber entsprechend langen varchar(...)} Datentypen liegt in der verbesserten Leistungsfähigkeit, da die Datenbanksysteme für den Zugriff vom Anwendungsprogramm auf die Datenbanksystem-LOBs spezielle Verfahren (sogenannte Locator) anbieten.

BLOB

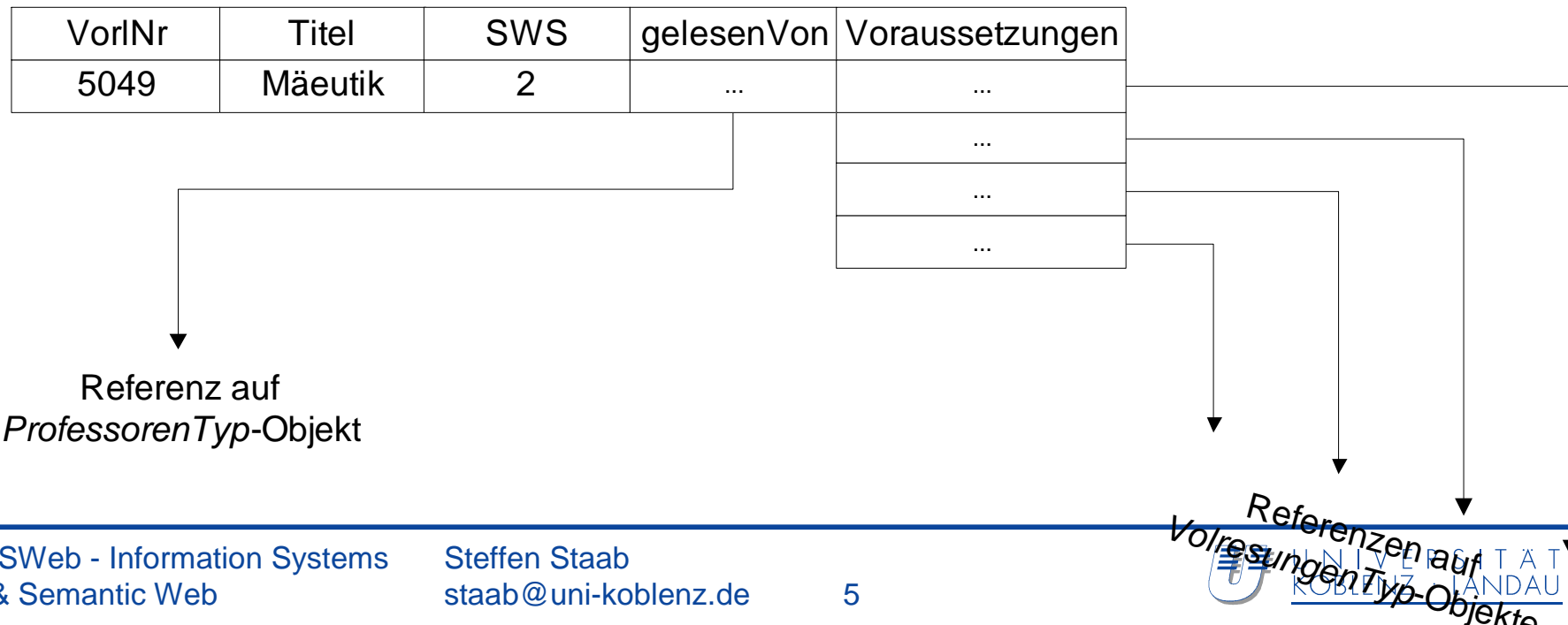
- ◆ In den Binary Large Objects speichert man solche Anwendungsdaten, die vom Datenbanksystem gar nicht interpretiert sondern nur gespeichert bzw.~archiviert werden sollen.

NCLOB

- ◆ CLOBs sind auf Texte mit 1-Byte Character-Daten beschränkt. Für die Speicherung von Texten mit Sonderzeichen, z.B.~Unicode-Texten müssen deshalb sogenannte National Character Large Objects (NCLOBs) verwendet werden
- ◆ In DB2 heißt dieser Datentyp (anders als im SQL:1999 Standard) DBCLOB -- als Abkürzung für Double Byte Character Large Object

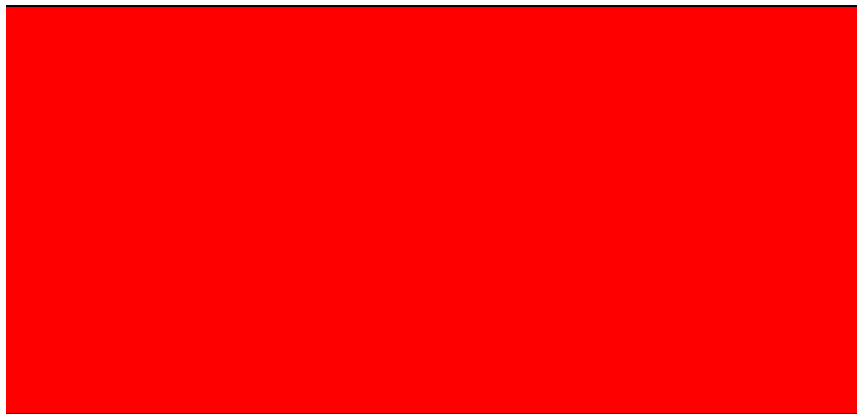
Mengenwertige Attribute

- ◆ Einem Tupel (Objekt) wird in einem Attribut eine Menge von Werten zugeordnet
- ◆ Damit ist es beispielsweise möglich, den Studenten ein mengenwertiges Attribut ProgrSprachenKenntnisse zuzuordnen.
- ◆ Schachtelung / Entschachtelung in der Anfragesprache



Geschachtelte Relationen

- ◆ Bei geschachtelten Relationen geht man noch einen Schritt weiter als bei mengenwertigen Attributen und erlaubt Attribute, die selbst wiederum Relationen sind.
- ◆ z.B. in einer Relation Studenten ein Attribut absolviertePrüfungen, unter dem die Menge von Prüfungen-Tupeln gespeichert ist.
- ◆ Jedes Tupel dieser geschachtelten Relation besteht selbst wieder aus Attributen, wie z.B. Note und Prüfer.



Typdeklarationen

- ◆ Objekt-relationale Datenbanksysteme unterstützen die Definition von anwendungsspezifischen Typen – oft user-defined types (UDTs) genannt.
- ◆ Oft unterscheidet man zwischen wert-basierten (Attribut-) und Objekt-Typen (Row-Typ).

Einfache Benutzer-definierte Typen: Distinct Types

```
CREATE DISTINCT TYPE NotenTyp AS DECIMAL (3,2) WITH  
COMPARISONS;
```

```
CREATE FUNCTION NotenDurchschnitt(NotenTyp) RETURNS NotenTyp  
Source avg(Decimal());
```

```
Create Table Pruefen (  
  MatrNr INT,  
  VorlNr INT,  
  PersNr INT,  
  Note NotenTyp);
```

```
Insert into Pruefen Values (28106,5001,2126,NotenTyp(1.00));
```

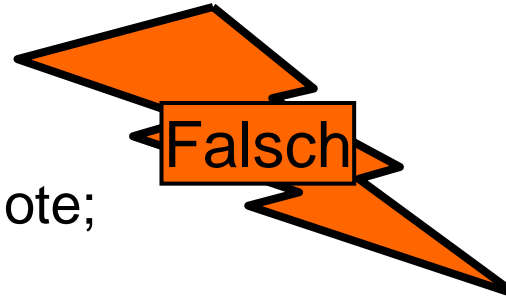
```
Insert into Pruefen Values (25403,5041,2125,NotenTyp(2.00));
```

```
Insert into Pruefen Values (27550,4630,2137,NotenTyp(2.00));
```

```
select NotenDurchschnitt(Note) as UniSchnitt from Pruefen;
```

Einfache Benutzer-definierte Typen: Distinct Types

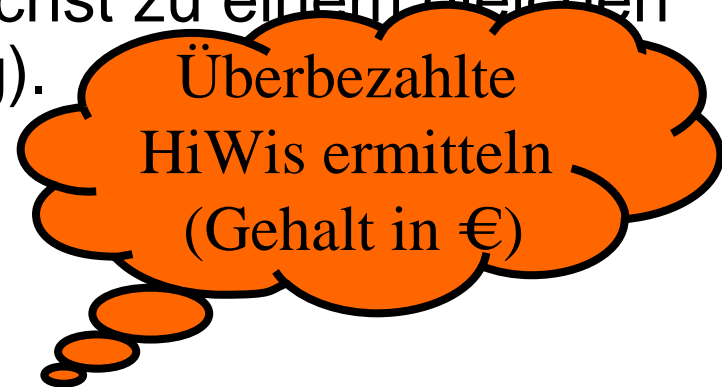
```
select *  
from Studenten s  
where s.Stundenlohn > s.VordiplomNote;
```



Geht nicht: Scheitert an dem unzulässigen Vergleich zweier unterschiedlicher Datentypen **NotenTyp vs. decimal**

Um unterschiedliche Datentypen miteinander zu vergleichen, muss man sie zunächst zu einem gleichen Datentyp transformieren (casting).

```
select *  
from Studenten s  
where s.Stundenlohn >  
      (9.99 - cast(s.VordiplomNote as decimal(3,2)));
```



```
CREATE DISTINCT TYPE US_NotenTyp AS DECIMAL (3,2) WITH  
COMPARISONS;
```

```
CREATE FUNCTION UsnachD_SQL(us US_NotenTyp) RETURNS  
NotenTyp
```

```
Return (case when Decimal(us) < 1.0 then NotenTyp(5.0)  
when Decimal(us) < 1.5 then NotenTyp(4.0)  
when Decimal(us) < 2.5 then NotenTyp(3.0)  
when Decimal(us) < 3.5 then NotenTyp(2.0)  
else NotenTyp(1.0) end);
```

```
Create Table TransferVonAmerika (  
MatrNr INT,  
VorlNr INT,  
Universitaet Varchar(30),  
Note US_NotenTyp);
```

Anwendung der Konvertierung in einer Anfrage<is web>

```
Insert into TransferVonAmerika Values (28106,5041,  
    'Univ. Southern California', US_NotenTyp(4.00));
```

```
select MatrNr, NotenDurchschnitt(Note)  
from  
    (  
        (select Note, MatrNr from Pruefen) union  
        (select USnachD_SQL(Note) as Note, MatrNr  
            from TransferVonAmerika)  
    ) as AllePruefungen  
group by MatrNr
```

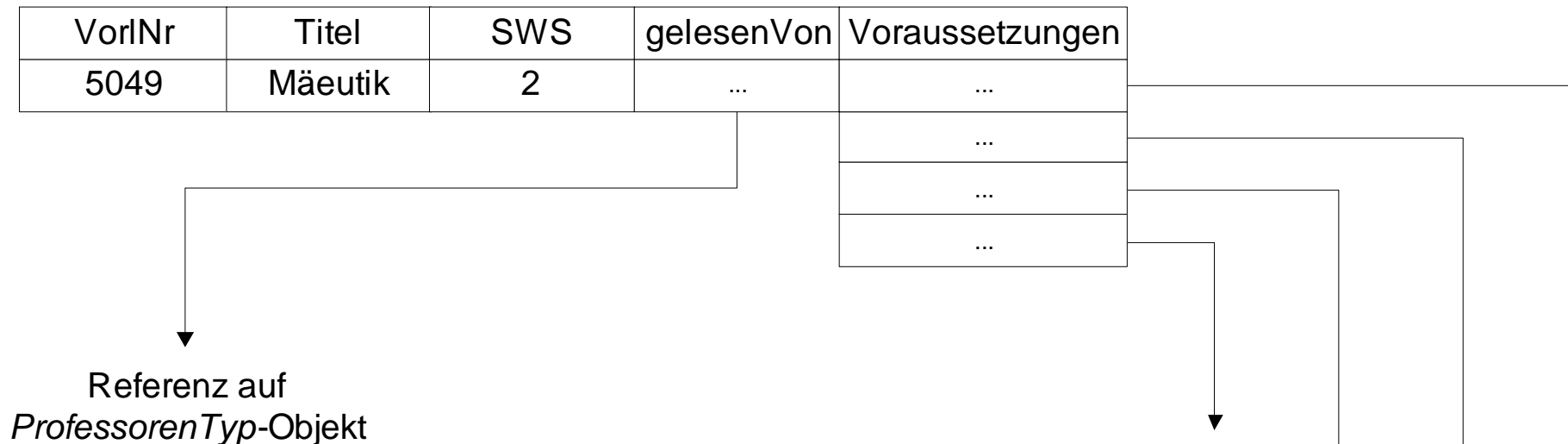
```
CREATE FUNCTION USnachD(DOUBLE) RETURNS Double
  EXTERNAL NAME 'Konverter_USnachD'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  FENCED;
```

```
CREATE FUNCTION UsnachD_Decimal (DECIMAL(3,2)) RETURNS
  DECIMAL(3,2)
  SOURCE USnachD (DOUBLE);
```

```
CREATE FUNCTION NotenTyp(US_NotenTyp) RETURNS NotenTyp
  SOURCE USnachD_Decimal (DECIMAL());
```

Referenzen

- ◆ Attribute können direkte Referenzen auf Tupel/Objekte (derselben oder anderer Relationen) als Wert haben.
- ◆ Dadurch ist man nicht mehr nur auf die Nutzung von Fremdschlüsseln zur Realisierung von Beziehungen beschränkt.
- ◆ Insbesondere kann ein Attribut auch eine Menge von Referenzen als Wert haben, so dass man auch N:M-Beziehungen ohne separate Beziehungsrelation repräsentieren kann
- ◆ Beispiel: Vorlesungen.gelesenVon ist Referenzen auf ProfessorTyp



Objektidentität

- ◆ Referenzen setzen natürlich voraus, dass man Objekte (Tupel) anhand einer unveränderlichen Objektidentität eindeutig identifizieren kann

Pfadausdrücke

- ◆ Referenzattribute führen unweigerlich zur Notwendigkeit, Pfadausdrücke in der Anfragesprache zu unterstützen.

Vererbung

- ◆ Die komplex strukturierten Typen können von einem Obertyp erben.
- ◆ Weiterhin kann man Relationen als Unterrelation einer Oberrelation definieren.
- ◆ Alle Tupel der Unter-Relation sind dann implizit auch in der Ober-Relation enthalten.
- ◆ Damit wird das Konzept der Generalisierung/Spezialisierung realisiert.

Operationen

- ◆ Den Objekttypen zugeordnet (oder auch nicht)
- ◆ Einfache Operationen können direkt in SQL implementiert werden
- ◆ Komplexere werden in einer Wirtssprache „extern“ realisiert
 - Java, C, PLSQL (Oracle-spezifisch), C++, etc.

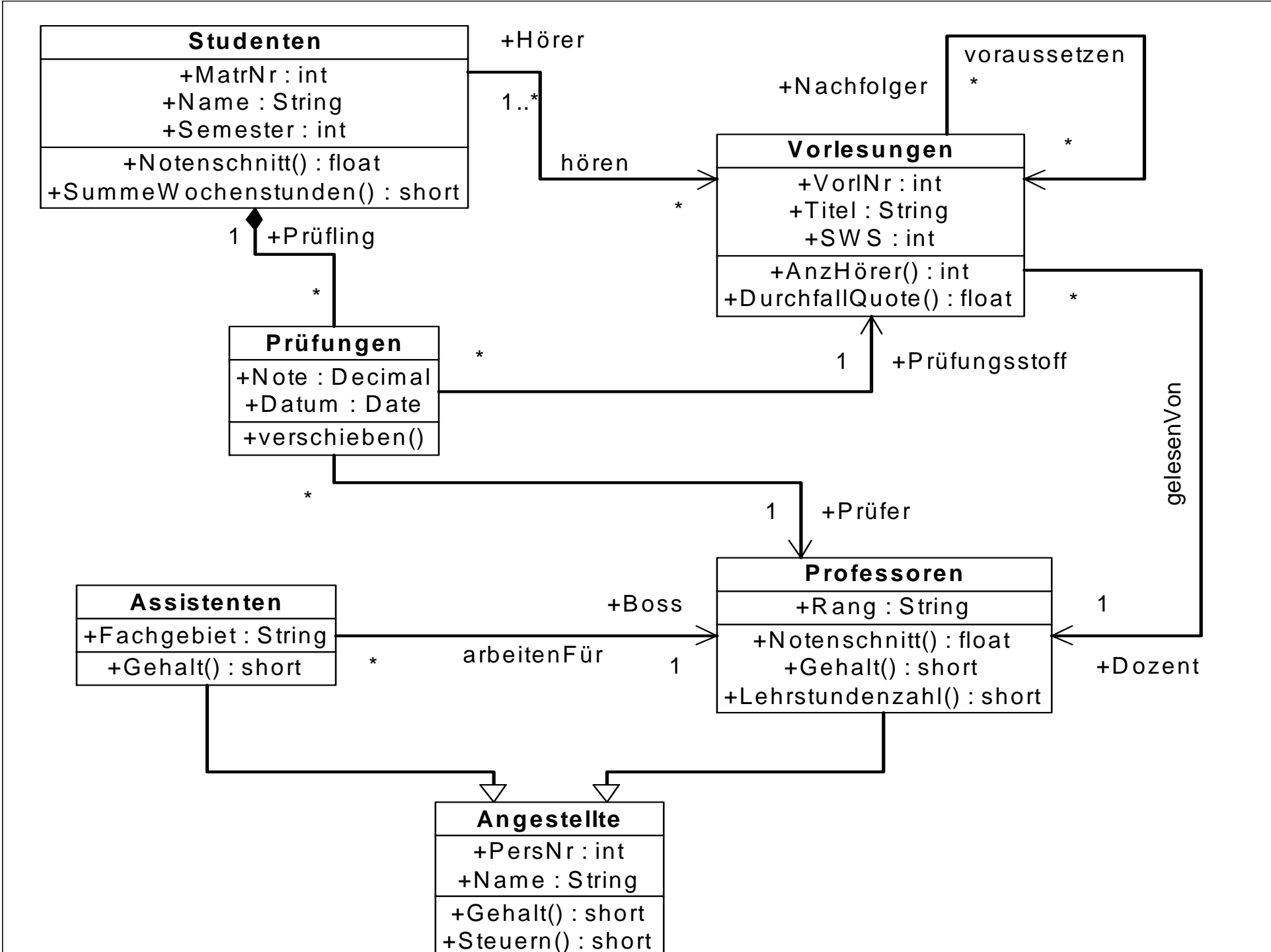
SQL2 bzw. SQL:1992

- ◆ Mindeststandard relationaler Datenbanksysteme
- ◆ Vorsicht: verschiedene Stufen der Einhaltung
 - Entry level ist die schwächste Stufe

SQL:1999

- ◆ Objekt-relationale Erweiterungen
- ◆ Trigger
- ◆ Stored Procedures
- ◆ Erweiterte Anfragesprache

Zunächst in UML



```
CREATE OR REPLACE TYPE ProfessorenTyp AS OBJECT (  
  PersNr NUMBER,  
  Name VARCHAR(20),  
  Rang CHAR(2),  
  Raum Number,  
  MEMBER FUNCTION Notenschnitt RETURN NUMBER,  
  MEMBER FUNCTION Gehalt RETURN NUMBER  
)
```

```
CREATE OR REPLACE TYPE AssistentenTyp AS OBJECT (  
  PersNr NUMBER,  
  Name VARCHAR(20),  
  Fachgebiet VARCHAR(20),  
  Boss REF ProfessorenTyp,  
  MEMBER FUNCTION Gehalt RETURN NUMBER  
)
```

```
CREATE OR REPLACE TYPE BODY ProfessorenTyp AS  
MEMBER FUNCTION Notenschnitt RETURN NUMBER is  
  BEGIN  
    /* Finde alle Prüfungen des/r Profs und  
       ermittle den Durchschnitt */  
  END;
```

```
MEMBER FUNCTION Gehalt RETURN NUMBER is  
  BEGIN  
    RETURN 1000.0; /* Einheitsgehalt für alle */  
  END;  
END;
```

```
CREATE TABLE ProfessorenTab OF ProfessorenTyp (  
    PersNr PRIMARY KEY) ;
```

```
CREATE TABLE AssistentenTab of AssistentenTyp;
```

```
INSERT INTO ProfessorenTab VALUES (2125, 'Sokrates', 'C4', 226);
```

```
INSERT INTO ProfessorenTab VALUES (2126, 'Russel', 'C4', 232);
```

```
INSERT INTO ProfessorenTab VALUES (2127, 'Kopernikus', 'C3', 310);
```

```
INSERT INTO ProfessorenTab VALUES (2133, 'Popper', 'C3', 52);
```

```
INSERT INTO ProfessorenTab VALUES (2134, 'Augustinus', 'C3', 309);
```

```
INSERT INTO ProfessorenTab VALUES (2136, 'Curie', 'C4', 36);
```

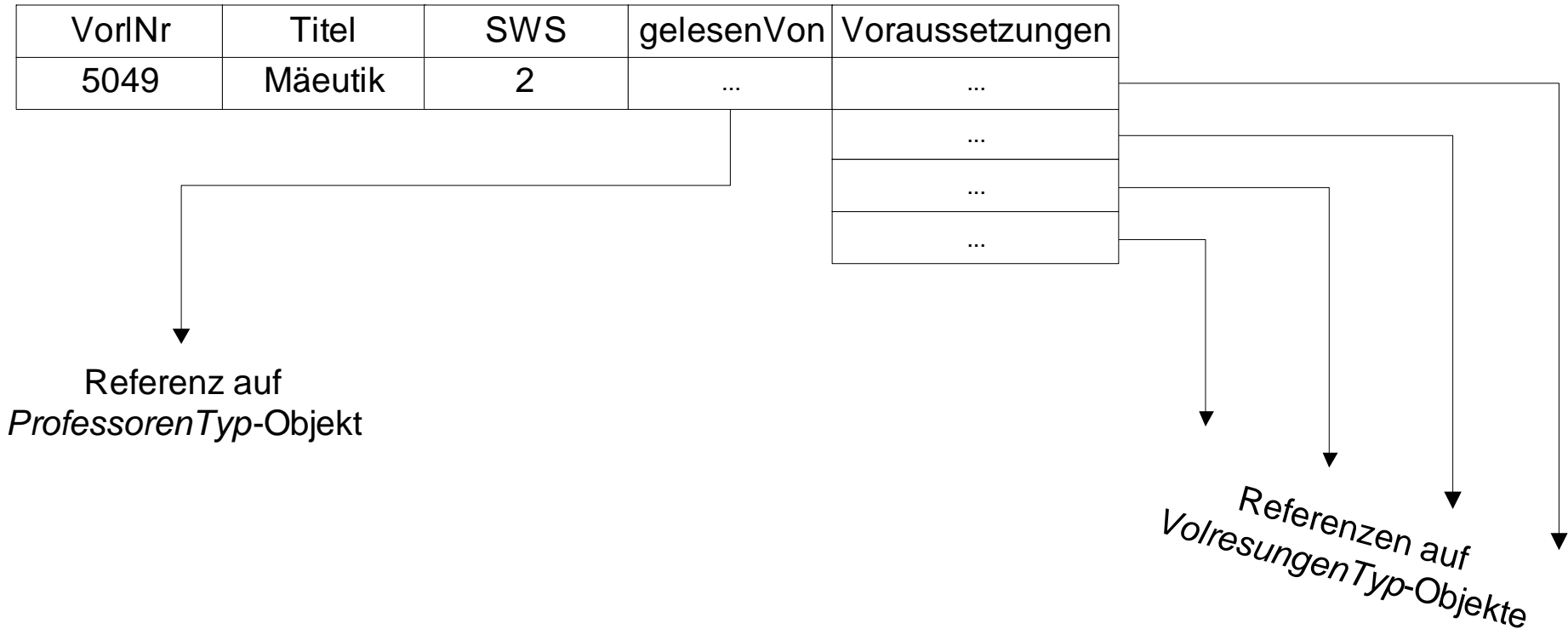
```
INSERT INTO ProfessorenTab VALUES (2137, 'Kant', 'C4', 7);
```

```
CREATE OR REPLACE TYPE VorlesungenTyp;
```

```
CREATE OR REPLACE TYPE VorlRefListenTyp AS TABLE OF REF  
    VorlesungenTyp
```

```
/
```

```
CREATE OR REPLACE TYPE VorlesungenTyp AS OBJECT (  
    VorlNr NUMBER,  
    TITEL VARCHAR(20),  
    SWS NUMBER,  
    gelesenVon REF ProfessorenTyp,  
    Voraussetzungen VorlRefListenTyp,  
    MEMBER FUNCTION DurchfallQuote RETURN NUMBER,  
    MEMBER FUNCTION AnzHoerer RETURN NUMBER  
)
```



```
CREATE TABLE VorlesungenTab OF VorlesungenTyp  
  NESTED TABLE Voraussetzungen STORE AS VorgaengerTab;
```

```
INSERT INTO VorlesungenTab
  SELECT 5041, 'Ethik', 4, REF(p), VorlRefListenTyp()
FROM ProfessorenTab p
WHERE Name = 'Sokrates';
```

```
insert into VorlesungenTab
  select 5216, 'Bioethik', 2, ref(p), VorlRefListenTyp()
from ProfessorenTab p
where Name = 'Russel';
```

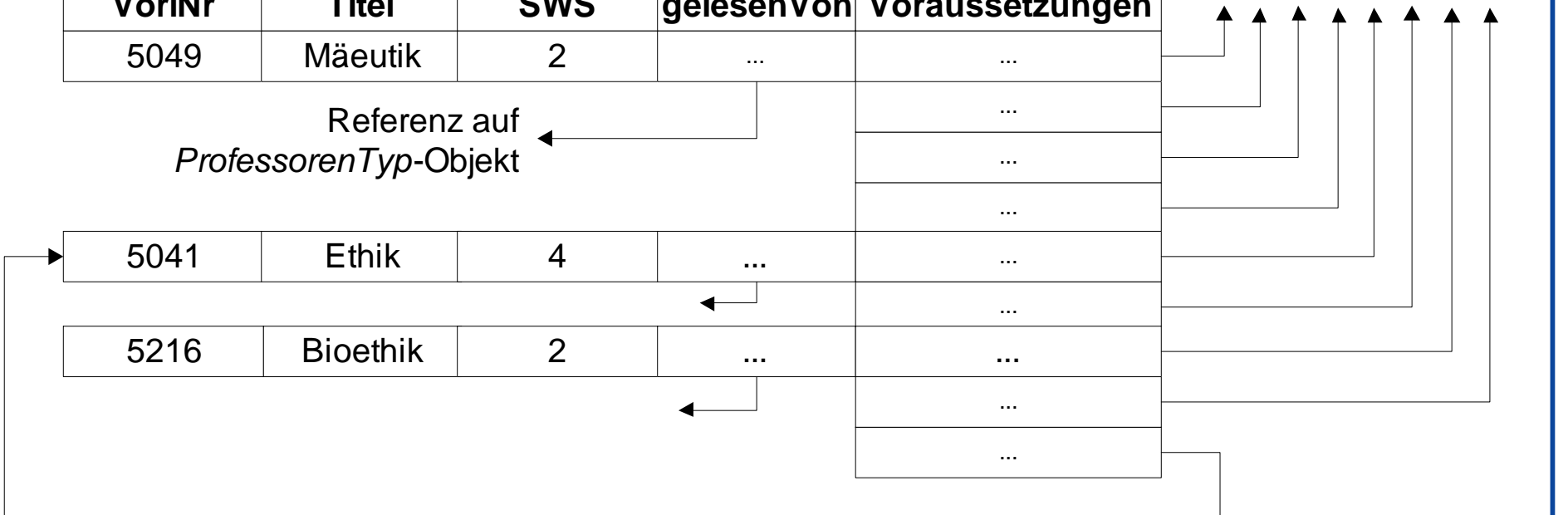
```
insert into table
  (select nachf.Voraussetzungen
   from VorlesungenTab nachf
   where nachf.Titel = 'Bioethik')
select ref(vorg)
from VorlesungenTab vorg
where vorg.Titel = 'Ethik';
```

VorlesungenTab

VorINr	Titel	SWS	gelesenVon	Voraussetzungen
5049	Mäeutik	2
				...
				...
				...
5041	Ethik	4
				...
5216	Bioethik	2
				...
				...

Referenz auf
ProfessorenTyp-Objekt

Referenzen auf
VorlesungenTyp-Objekte



```
CREATE TYPE Angestellte_t AS  
  (PersNr INT,  
   Name VARCHAR(20))  
INSTANTIABLE  
REF USING VARCHAR(13) FOR BIT DATA  
MODE DB2SQL;
```

```
CREATE TYPE Professoren_t UNDER Angestellte_t AS  
  (Rang CHAR(2),  
   Raum INT)  
MODE DB2SQL;
```

```
CREATE TYPE Assistenten_t UNDER Angestellte_t AS  
  (Fachgebiet VARCHAR(20),  
   Boss REF(Professoren_t))  
MODE DB2SQL;
```

```
ALTER TYPE Professoren_t  
  ADD METHOD anzMitarb()  
  RETURNS INT  
  LANGUAGE SQL  
  CONTAINS SQL  
  READS SQL DATA;
```

```
CREATE TABLE Angestellte OF Angestellte_t  
  (REF IS Oid USER GENERATED);
```

```
CREATE TABLE Professoren OF Professoren_t UNDER  
  Angestellte  
  INHERIT SELECT PRIVILEGES;
```

```
CREATE TABLE Assistenten OF Assistenten_t UNDER Angestellte  
  INHERIT SELECT PRIVILEGES  
  (Boss WITH OPTIONS SCOPE Professoren);
```

```
CREATE METHOD anzMitarb()  
  FOR Professoren_t  
  RETURN (SELECT COUNT (*)  
          From Assistenten  
          WHERE Boss->PersNr = SELF..PersNr);  
  
INSERT INTO Professoren (Oid, PersNr, Name, Rang, Raum)  
  VALUES(Professoren_t('s'), 2125, 'Sokrates', 'C4', 226);  
  
INSERT INTO Professoren (Oid, PersNr, Name, Rang, Raum)  
  VALUES(Professoren_t('r'), 2126, 'Russel', 'C4', 232);  
  
INSERT INTO Professoren (Oid, PersNr, Name, Rang, Raum)  
  VALUES(Professoren_t('c'), 2137, 'Curie', 'C4', 7);  
  
INSERT INTO Assistenten (Oid, PersNr, Name, Fachgebiet, Boss)  
  VALUES(Assistenten_t('p'), 3002, 'Platon', 'Ideenlehre',  
  Professoren_t('s'));
```

```
INSERT INTO Assistenten (Oid, PersNr, Name, Fachgebiet, Boss)
VALUES(Assistenten_t('a'), 3003, 'Aristoteles', 'Syllogistik',
Professoren_t('s'));
```

```
INSERT INTO Assistenten (Oid, PersNr, Name, Fachgebiet, Boss)
VALUES(Assistenten_t('w'), 3004, 'Wittgenstein',
'Sprachtheorie', Professoren_t('r'));
```

```
select a.name, a.PersNr
from Angestellte a;
```

```
select * from Assistenten;
```

```
select a.Name, a.Boss->Name, a.Boss->wieHart() as Güte
from Assistenten a;
```