

# 3. Datenbanksprachen, SQL

## 3.1. Datenbankdefinition

Grundzüge:

Grundstruktur: Tabelle (Relation) mit atomaren Attributen

Attributen werden Datentypen zugeordnet

Werte können auch Nullwert oder default sein

evt. Repräsentationstyp

Zusammenfassung in Schema

**Schema**      CREATE SCHEMA [SchemaName] [AUTHORIZATION BenName]  
                  [DEFAULT CHARACTER SET characterListe]  
                  [ ListeTabellenDef]

**Tabelle**      CREATE TABLE Name (AttrDefListe    TabIntegritätsbedingListe);  
  
                  CREATE TEMPORARY TABLE ....

**AttrDefinition**    AttrName    Datentyp | Domain [DefaultDef] [IntegritätsbedingListe]

**Datentypen:** NATIONAL CHAR[ACTER] [VARYING] | VARCHAR |  
INT[EGER] | SMALLINT | NUMERIC | DEC[IMAL] | FLOAT |  
REAL | DOUBLE PRECISION | BIT | BIT VARYING | DATE |  
TIME [WITH TIME ZONE] | TIMESTAMP [WITH TIME ZONE] |  
INTERVAL

‘Love story’, 2345, 23, 2.99, 1.99, 1.56E-4, 1.56E-4, 3.1415929E00, B‘0110110’,  
DATE‘1998-12-14’, TIME‘17:15:00.05’, TIMESTAMP‘1998-12-14 17:15:00.05’,  
TIMESTAMP‘1998-12-14 17:15:00.05+01:00’,  
INTERVAL‘6’DAY \* 3

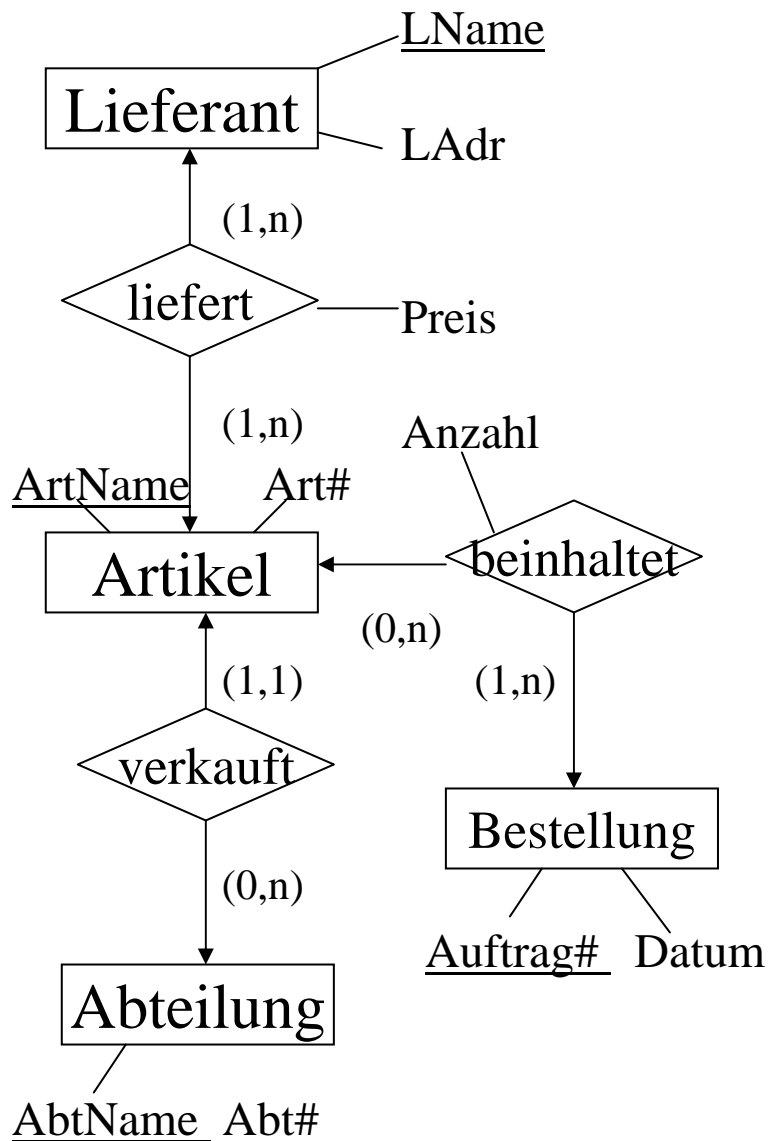
SQL ist streng typisiert

deshalb ist eine Typumwandlung erforderlich

x : DECIMAL(6,2); y : REAL  
CAST ( x AS REAL) + y < CAST (5 AS REAL)

**Löschen:** DROP TABLE TabellenName

**Verändern:** ALTER TABLE TabellenName ADD AttrDefinition



LIEFERANT(LName, LAdr)  
 ABTEILUNG(AbtName, Abt#)

ARTIKEL(ArtName, Art#, verkauftAbtName)  
 BESTELLUNG(Auftrag#, Datum)  
**liefert**(LName, ArtName, Preis)  
 beinhaltet(Auftrag#, ArtName, Anzahl)

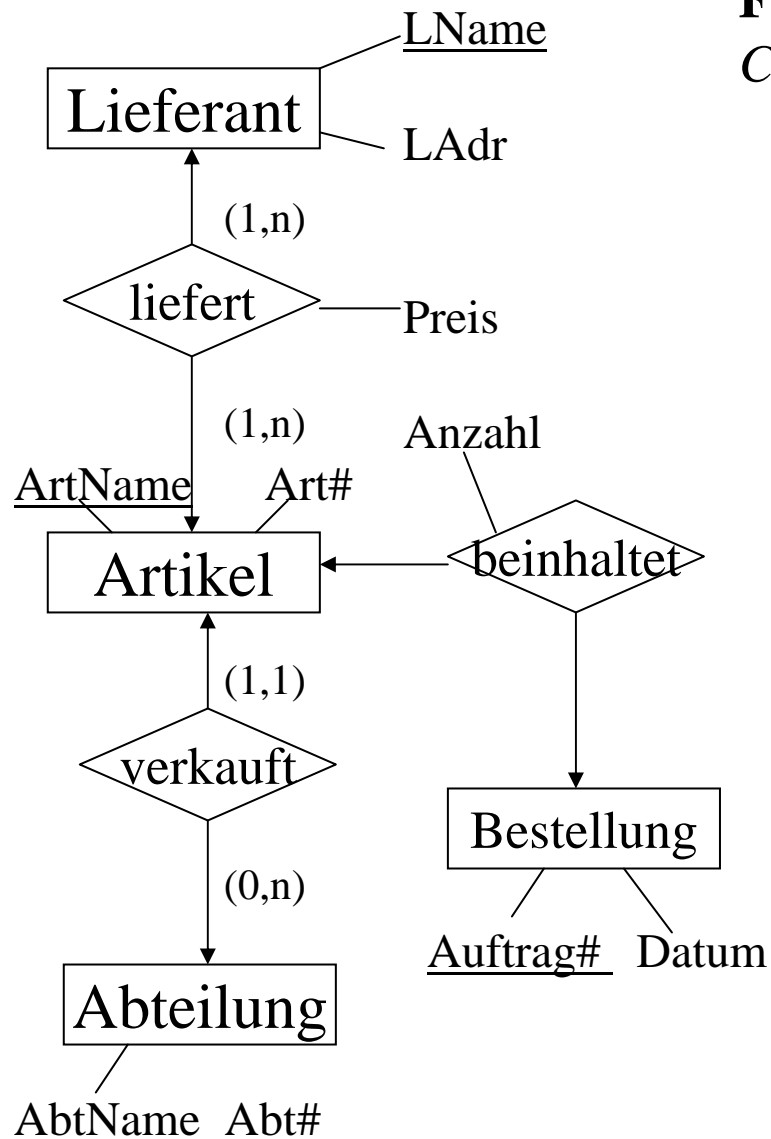
```

CREATE TABLE liefert
  (LName VARCHAR(20),
   ArtName VARCHAR(10),
   Preis DECIMAL(6,2),
   CONSTRAINT liefert_pk
     PRIMARY KEY (LName,ArtName),
   ....
   ..... );
  
```

Darstellung der **Schlüsselbedingung**  
*CONSTRAINT name PRIMARY KEY (attrListe)*  
*CONSTRAINT name UNIQUE(attrListe)*

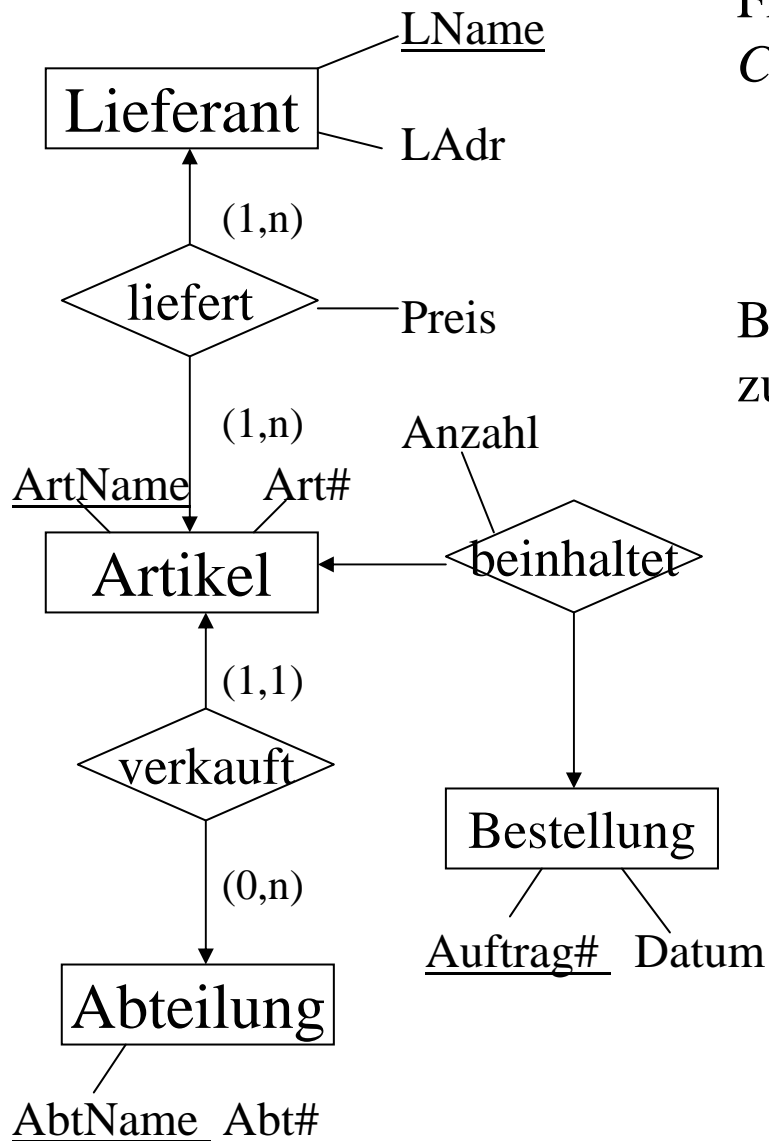
## Darstellung der Kardinalitätsbedingung als Fremdschlüssel

*CONSTRAINT name FOREIGN KEY (attrListe)  
REFERENCES tabelle (attrListe) :*



LIEFERANT(LName, LAdr)  
ARTIKEL(ArtName, Art#)  
liefert(LName, ArtName, Preis)

```
CREATE TABLE liefert
(LName VARCHAR(20),
 ArtName VARCHAR(10),
 Preis DECIMAL(6,2),
 CONSTRAINT liefert_pk
 PRIMARY KEY (LName, ArtName),
 CONSTRAINT liefert_fk1
 FOREIGN KEY (LName)
 REFERENCES LIEFERANT,
 CONSTRAINT liefert_fk2
 FOREIGN KEY (ArtName)
 REFERENCES ARTIKEL2);
```



Darstellung der Kardinalitätsbedingung als Fremdschlüssel mit referentiellen Aktionen  
*CONSTRAINT name FOREIGN KEY (attrListe)*  
*REFERENCES tabelle (attrListe)*  
 aktionen:

Bei Verletzung der Integrität werden Aktionen zur Kompensation ausgelöst.

- DEFAULT-Aktion: Setzen auf Default-Wert
- NULL-Aktion: Setzen auf Nullwert (wenn Fremdschlüssel nicht Primärschlüssel)
- CASCADE-Aktion: Fortsetzung auf Relation

```

CREATE TABLE liefert
(LName VARCHAR(20)
  DEFAULT 'niemand',
  ....
  CONSTRAINT liefert_fk1
  FOREIGN KEY (LName)
  REFERENCES LIEFERANT
  ON DELETE SET DEFAULT,
  ....);
  
```

**Referentielle Integrität**  $R[X] \subseteq T[Y]$  (Y Schlüssel in T)  
 bedingt Auslösung von Aktionen auf referenzierender Relation R,  
 wenn eine Aktion auf referenzierter Relation T

	in R	in T
Einfügen	fehlende Referenz: zurückweisen	kein Problem
Löschen	kein Problem	<i>Problembhebung</i>
Modifikation	fehlende Referenz: zurückweisen	<i>Problembhebung</i>

	beinhaltet			Artikel			
	AuftragNr	ArtNr	Anzahl	ArtNr	Bez	Preis	
	1	1	7	1	a	1	
	1	3	6	2	b	15	DELETE ???, UPDATE ArtNr??
	1	2	3	3	a	16	
Insert nicht erlaubt	<del>1</del>	<del>4</del>	<del>1</del>	5	c	2	DELETE ok

**Integritätsbedingungen** PRIMARY KEY | UNIQUE (Attributliste) |  
 FOREIGN KEY (Attributliste) REFERENCES TabName [(Attributliste)]  
 [ MATCH FULL | MATCH PARTIAL ]  
 [ ON DELETE NO ACTION | CASCADE | SET DEFAULT |  
 SET NULL ]  
 [ ON UPDATE NO ACTION | CASCADE | SET DEFAULT |  
 SET NULL ]  
 [CHECK Klausel]  
 [ASSERTION Bedingung]

## Beispiel: Datentyp-Integritätsbedingung

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

$\forall m \in \text{Mitarbeiter:}$

$(m.\text{Extern} \in \{\text{FALSE}, \text{TRUE}\})$

### Datenbanksystem: ORACLE

CREATE TABLE Mitarbeiter

```
(PersNr      CHAR (10) NOT NULL,
 Name        CHAR (20) NOT NULL,
 Vorname     CHAR (20) NOT NULL,
 Extern      CHAR (5)  NOT NULL
             CHECK (Extern = „TRUE“ OR
                   Extern = „FALSE“),
```

```
Adresse      CHAR (120),
TNr          CHAR (20),
Urlaubsanspruch INTEGER,
Resturlaub   INTEGER,
Sonderurlaub INTEGER,
Institut     CHAR (40));
```

### Datenbanksystem: SYBASE

SP\_ADDTYPE **bool**, „CHAR (5)“, NOT NULL

CREATE RULE **TrueFalse** AS

@Wert IN („TRUE“, „FALSE“);

SP\_BINDRULE **TrueFalse**, **bool**;

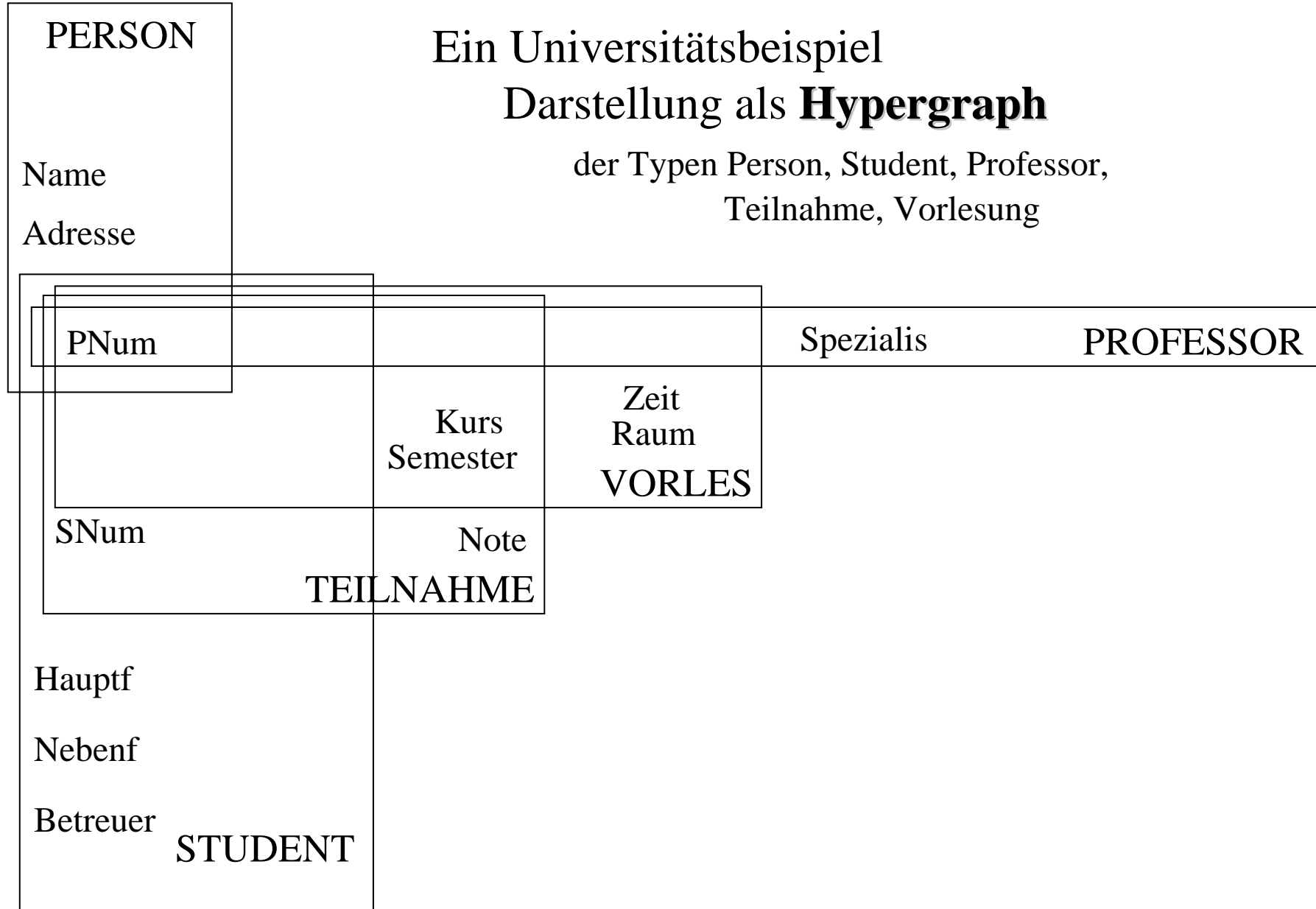
CREATE TABLE Mitarbeiter

```
(PersNr      CHAR (10) NOT NULL,
 Name        CHAR (20) NOT NULL,
 Vorname     CHAR (20) NOT NULL,
 Extern      bool,
 Adresse     CHAR (120),
TNr          CHAR (20),
Urlaubsanspruch INTEGER,
Resturlaub   INTEGER,
Sonderurlaub INTEGER,
Institut     CHAR (40));
```

# Ein Universitätsbeispiel

## Darstellung als **Hypergraph**

der Typen Person, Student, Professor,  
Teilnahme, Vorlesung



# SQL-Definition des Universitätsschemas

```
CREATE SCHEMA
```

```
UniEinfach AUTHORIZATION Spur (
```

```
CREATE TABLE PERSON
```

```
(Name VARCHAR (15) NOT NULL,
```

```
Adress VARCHAR (30),
```

```
PNum INTEGER,
```

```
PRIMARY KEY (PNum));
```

```
CREATE TABLE STUDENT
```

```
(PNum INTEGER NOT NULL,
```

```
SNum INTEGER NOT NULL,
```

```
Hauptf CHAR(3) NOT NULL,
```

```
Nebenf CHAR(3),
```

```
Betreuer VARCHAR (15),
```

```
PRIMARY KEY (SNum),
```

```
FOREIGN KEY (PNum)
```

```
REFERENCES PERSON);
```

```
CREATE TABLE PROFESSOR
```

```
(PNum INTEGER NOT NULL,
```

```
Spezialis CHAR (10),
```

```
PRIMARY KEY (PNum),
```

```
FOREIGN KEY (PNum)
```

```
REFERENCES PERSON);
```

```
CREATE TABLE VORLES
```

```
(Kurs VARCHAR (20) NOT NULL,
```

```
Raum CHAR (5),
```

```
Zeit CHAR (3),
```

```
Semester CHAR (5) NOT NULL,
```

```
LesenderPNum INTEGER NOT NULL,
```

```
PRIMARY KEY (Raum, Zeit, Semester),
```

```
FOREIGN KEY (LesenderPNum)
```

```
REFERENCES PROFESSOR(PNum));
```

```
CREATE TABLE TEILNAHME
```

```
(Kurs VARCHAR (20) NOT NULL,
```

```
Semester CHAR (5) NOT NULL,
```

```
Lesender PNum INTEGER NOT NULL,
```

```
SNum INTEGER,
```

```
Note DEC (2,1),
```

```
PRIMARY KEY (SNum, Kurs, Semester),
```

```
FOREIGN KEY (SNum) REFERENCES STUDENT,
```

```
NOT NULL ein Student muß existieren
```

```
ON DELETE RESTRICT Nichtentfernen des
```

```
letzten Student
```

```
ON UPDATE CASCADE)
```

```
)
```

```
DROP TABLE TEILNAHME
```

```
ALTER TABLE VORLES ADD SNum INTEGER
```

## Sichtendefinition

als virtuelle Tabelle

definiert über Anfrage an die Datenbank

```
CREATE VIEW ViewName [(ListeViewAttributName)]  
AS Anfrage
```

mit gleicher Anzahl von Attributen

Löschen der Sicht: DROP VIEW ViewName

Vorsicht mit Updates über Sichten

## Indizes in SQL

für die effiziente Verarbeitung und Suche

```
CREATE [UNIQUE] INDEX IndexName  
ON Tabellename (AttrListe)
```

## Zugriffsrechte

```
GRANT {ALL PRIVILEGES | {SELECT | DELETE | INSERT | UPDATE |  
REFERENCES} (AttrListe) ON TABLE TabName } [...]  
TO {PUBLIC | authIDListe} [WITH GRANT OPTION]
```

*Beispiele:*

INSTITUT		
Bezeichnung	String	KEY
Adresse	String	
PersNrLeiter	String	

```
CREATE TABLE Institut
  (Bezeichnung CHAR (40) NOT NULL,
  Adresse CHAR (120),
  PersNrLeiter CHAR (10) NOT NULL);
```

→

```
CREATE UNIQUE INDEX InstitutKEY ON
  Institut (Bezeichnung);
```

```
CREATE INDEX InstitutLeiter ON
  Institut (PersNrLeiter);
```

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

→

```
CREATE TABLE Mitarbeiter
(PersNr      CHAR (10) NOT NULL,
Name        CHAR (20) NOT NULL,
Vorname     CHAR (20) NOT NULL,
Extern      CHAR (5)  NOT NULL,
Adresse     CHAR (120),
TNr         CHAR (20),
Urlaubsanspruch  INTEGER,
Resturlaub  INTEGER,
Sonderurlaub INTEGER,
Institut    CHAR (40));
```

```
CREATE UNIQUE INDEX MitarbeiterKEY ON
Mitarbeiter (PersNr);
```

```
CREATE INDEX MitarbeiterName ON
Mitarbeiter (Name);
```

```
CREATE INDEX MitarbeiterInstitut ON
Mitarbeiter (Institut);
```

Urlaubsliste		
PersNr	string	KEY
Anfang	date	KEY
Ende	date	
genommen	bool	
Sonderurlaub	bool	
AnzahlTage	int	

→

CREATE TABLE Urlaubsliste

(PersNr CHAR (10) NOT NULL,  
Anfang DATE NOT NULL,  
Ende DATE NOT NULL,  
genommen CHAR (5) NOT NULL,  
Sonderurlaub CHAR (5) NOT NULL,  
AnzahlTage INTEGER NOT NULL;

CREATE UNIQUE INDEX UrlaubslisteKEY ON  
Urlaubsliste (PersNr, Anfang);

CREATE INDEX UrlaubslistePersNr ON  
Urlaubsliste (PersNr);

## 3.2. Modifikationsoperationen (Ein-Tabellen-Operationen)

**Einfügen:** Syntax INSERT INTO Person VALUES ('Gerste',KI,007)  
INSERT INTO Professor  
*anfrage*

Semantik: Einfügen in eine (Multi-)menge  
für Schemata: CREATE

**Modifizieren:** Syntax: UPDATE Vorles  
SET Raum = "CAP3, R1"  
WHERE Kurs = "Datenbanken" AND ...

Semantik: i.a. Mehrtupeloperation; Select in WHERE mit Einschränkungen  
für Schemata: ALTER (z.B. auch ALTER TABLE DROP COLUMN ...)

**Löschen:** Syntax: DELETE FROM Vorlesung WHERE Kurs = "Datenbanken"  
Semantik: Streichen von Tupeln (nicht nur Attributwerten)

für Schemata: DROP

⊗ Die Mitarbeiterin Anja Meier hat ihren Nachnamen in Meier-Schulz geändert.

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
UPDATE Mitarbeiter
    SET Name = "Meier-Schulz"
WHERE Name = "Meier"
    AND Vorname = "Anja";
```

⊗ Alle Mitarbeiter des Instituts „Grundlagen der Agrartechnik“ dürfen ihren Resturlaub auch nach dem 31. März antreten.

Mitarbeiter		
PersNr	string	KEY
Name	string	
Vorname	string	
Extern	bool	
Adresse	string	
TNr	string	
Urlaubsanspruch	int	
Resturlaub	int	
Sonderurlaub	int	
Institut	string	

```
UPDATE Mitarbeiter
    SET Urlaubsanspruch =
        Urlaubsanspruch + Resturlaub
    WHERE Institut =
        „Grundlagen der Agrartechnik“
```

```
UPDATE Mitarbeiter
    SET Resturlaub = 0
    WHERE Institut =
        „Grundlagen Agrartechnik“;
```

⊗ Für den Kurs mit der Nummer „IT/17“ haben sich zuwenig Teilnehmer angemeldet. Sie sollen auf den Kurs „IT/15“ umgebucht werden. Alle Daten über den Kurs „IT/17“ sollen gelöscht werden .

Kurs		
KursNr	string	KEY
Titel	string	
Inhalt	string	
max Teilnehmer	int	
Endtermin	date	
Leitung	string	

Voraussetzung		
KursNr	string	KEY
VoraussNr	string	KEY

Kursteilnahme		
KursNr	string	KEY
PersNr	string	KEY

Kursdurchführung		
KursNr	string	KEY
Datum	date	KEY
Anfang	time	KEY
Ende	time	KEY
Ort	string	

```
DELETE FROM Kurs
      WHERE KursNr = "IT/17";
```

```
DELETE FROM Voraussetzung
      WHERE KursNr = "IT/17";
```

```
DELETE FROM Kursdurchführung
      WHERE KursNr = "IT/17";
```

```
UPDATE Kursteilnahme
      SET KursNr = "IT/15"
      WHERE KursNr = "IT/17";
```

## 3.3. Transaktionen in DBS (Parallelität von Anweisungen)

### Zuverlässigkeit von Datenbankanwendungen

1. Datenbankoperationen bestehen aus vielen zusammenhängenden Operationen

Problem: Rechnerabsturz inmitten einer Folge

Lösung: entweder **vollständig** oder **gar nicht** (*atomicity*)

2. Integritätsbedingungen widersprechen den durchgeführten Änderungen  $\Rightarrow$  rollback

Lösung: Pflege der **Konsistenz** (*consistency*)

3. Parallele Ausführung von Datenbankoperationen durch verschiedene Benutzer

A fragt nach allen freien Zimmern in Berlin am x. sowie Flügen am x+1;

B fragt ebenso für x.; B bucht letztes Zimmer; A bucht Flug; A will Hotel buchen;

Lösung: Datenbankoperationen verschiedener Benutzer **isoliert** (*isolated*)

4. Technischer Defekt - was wird dann mit den Daten?

Lösung: Vollständig ausgeführte Operationen bleiben **erhalten** (*durability*)

Generelle Lösung:

Zusammenfassung von Operationen zu **Transaktionen**,

die atomar, konsistent, isoliert, dauerhaft sind (ACID)

$\Rightarrow$  1. Sprachkonstrukt

**Begin\_of\_transaction** op1; ...; opn; **End\_of\_transaction**

2. Umsetzung in DBMS

## Realisierungsideen:

**A. Transaktionen werden explizit verwaltet und erhalten Identifikation**

**B. Erzeugen eines Zeitpunktes und sicheres Zwischenspeichern eines Zustandes**

1. Problem: Atomarität von Operationen: Änderungen erst bei End\_of\_transaction in die DB geschrieben; Erzeugen von Rücksetzinformationen

4. Problem: Erhaltung von Änderungen: alle abgeschlossenen Transaktionen werden protokolliert (Log); nach Fehler nochmalige Durchführung

**C. Verwaltung von Ressourcen und exklusive Zuordnung**

3. Problem: Transaktionen sperren Objekte in 2 Phasen (Anforderung von allen Sperren; Bearbeitung; Freigabe aller Sperren am Ende der Transaktion)

Neues Problem: *deadlock* (A sperrt Relation Hotel; B sperrt Flüge; A will nun Flüge sperren und muß auf B warten; B will Hotel sperren und muß auf A warten)

Lösung: Zurückfahren des Systems; Ausführung anderer oder einer TA

Weiteres Problem: zu große Teile der Datenbank gesperrt (Granularität)

Lösung: unterschiedliche Granulate, Einschränkung der Isolierung (siehe nächste Folie)

**D. Explizite Bestätigung der Korrektheit der Transaktion**

2. Problem: vor EOT wird Korrektheit geprüft; evt. Zurücksetzen auf Anfangszustand (rollback) (evt. teilweises Zurücksetzen bis zu *safepoint*)

(Zustände: *BOT*, *EOT*, *failed*, *commit*, *partial\_commit*)

# Probleme bei Einschränkung der Isolierung

Isolierung garantiert Konsistenz

Realisierung muß alle möglichen Operationen in Betracht ziehen  
damit Einschränkung der Parallelisierung

aber je nach Anwendung: nur bestimmte Operationen genutzt

Konsistenz vielleicht nicht erforderlich

damit unnötige Einschränkung der Parallelität

deshalb feinere skalierbare Sperrverfahren, anderes TA-Konzept

Einschränkung der Isolierung kann Probleme bringen

*Dirty read*

TA1	TA2
BOT	
insert x	
	read x
Rollback	
Element war nie vorhanden	

*Nichtwiederholbares  
read*

TA1	TA2
BOT	
read x	
	BOT
	delete x
	COMMIT
read x	
2 unterschiedl. Zustände	

*Verlorenes  
update*

TA1	TA2
BOT	
	BOT
read x	
x=x+n	
	read x
write x	x=x-m
	write x
COMMIT	

*Phantome*

(Veränderung des  
Inhaltes während/  
zwischen Anfragen)

*Temporärer Update*

(bei fail)

*Falsche Summe*

(Wert in Veränd.)

FAIL oder COMMT

Lösung: **Scheduling** (Planung der Reihenfolge, Serialisierung)