

---

# Grundlagen von Datenbanken

---

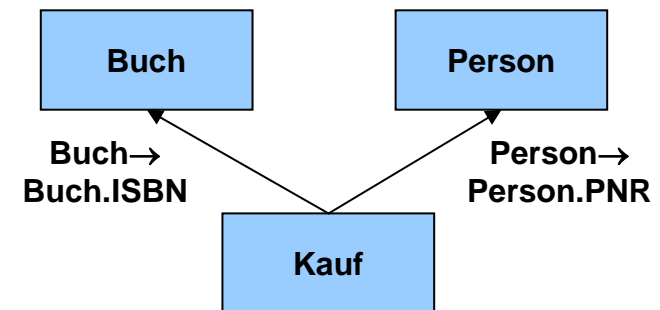
## Aufgabenzettel 5

referentielle Aktionen, Sichten, Serialisierbarkeit und  
Locking



# SQL DDL: Referentielle Aktionen (1/3)

- Potentielle Gefährdung der referentiellen Integrität durch Änderungsoperationen
- Referentielle Integrität:  
Einfügen/Ändern von FS-Attributen an der Sohn-Relation (*Kauf*)
  - Prüfung ob PS zu FS-Wert vorhanden ist
  - Operation wird verhindert falls referentielle Integrität verletzt
- Reaktion auf Einfügen/Ändern/Löschen an den Vater-Relationen (*Person/Buch*) falls abhängige Tupel in der Sohn-Relation existieren
  - Operation verbieten?
  - Tupel rekursiv ändern/löschen?
  - FS-Wert in der Sohn-Relation auf NULL setzen?



# SQL DDL: Referentielle Aktionen (2/3)

## ■ Syntax

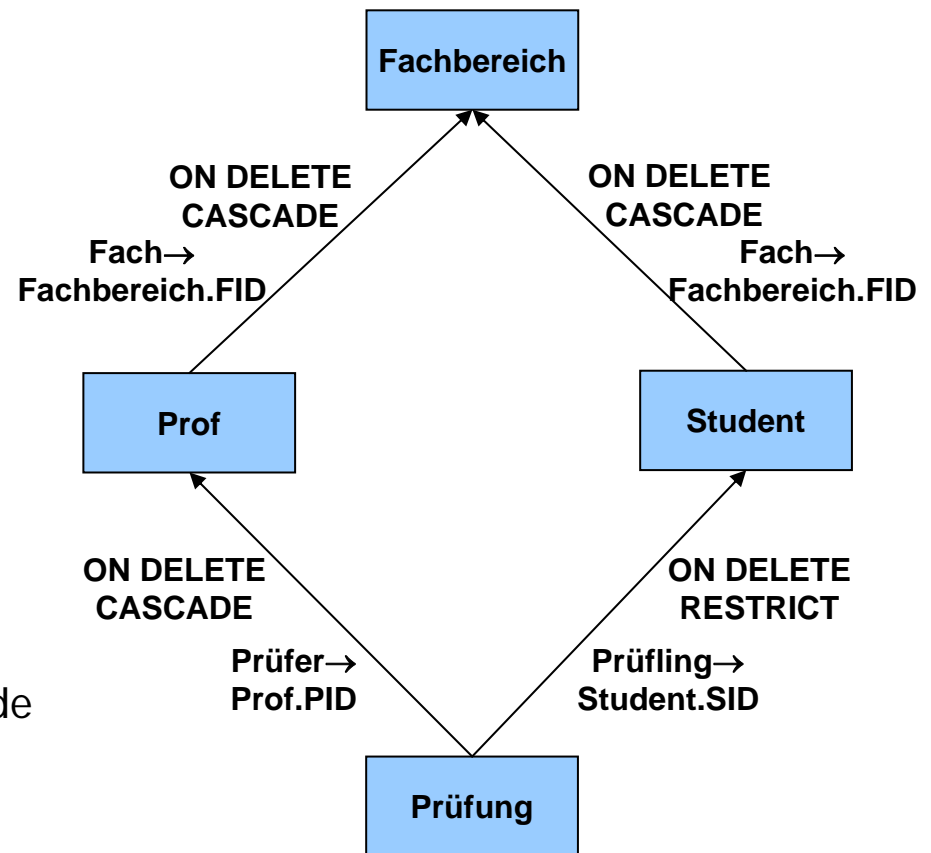
```
<referential triggered action> ::=  
    <update rule> [ <delete rule> ] | <delete rule> [ <update rule> ]  
  
<update rule> ::= ON UPDATE <referential action>  
<delete rule> ::= ON DELETE <referential action>  
  
<referential action> ::=  
    CASCADE | SET NULL | SET DEFAULT | NO ACTION | RESTRICT
```

## ■ Beispiel

```
CREATE TABLE Kauf (  
    Person      int,  
    Buch        varchar(13),  
    CONSTRAINT pk_kauf PRIMARY KEY (Person, Buch),  
    CONSTRAINT fk_pers FOREIGN KEY (Person)  
        REFERENCES Person(PNR) ON DELETE CASCADE ON UPDATE CASCADE,  
    CONSTRAINT fk_buch FOREIGN KEY (Buch)  
        REFERENCES Buch (ISBN) ON DELETE RESTRICT ON UPDATE RESTRICT);
```

# SQL DDL: Referentielle Aktionen (3/3)

- Löschen eines Fachbereichs
- "erst links":
  - Löschen in FB
  - Löschen in PROF
  - Löschen in PRUEFUNG
  - Löschen in STUDENT
  - Zugriff auf PRÜFUNG: Wenn ein Student bei einem FB-fremden Professor geprüft wurde → Rücksetzen
- "erst rechts":
  - Löschen in FB
  - Löschen in STUDENT
  - Zugriff auf PRÜFUNG: Wenn ein gerade gelöschter Student eine Prüfung abgelegt hatte → Rücksetzen
  - Löschen in PROF
  - Löschen in PRUEFUNG



**Reihenfolge-abhängiges Ergebnis  
⇒ kein sicheres Schema**

# Sichtendefinition

## **SQL DDL**

```
CREATE TABLE Student (  
    MNR          int          PRIMARY KEY,  
    Vorname      varchar(50),  
    Nachname     varchar(50) NOT NULL,  
    Fach         varchar(50) NOT NULL,  
    Studiengang  varchar(50) NOT NULL,  
    Semester     int          NOT NULL,  
)
```

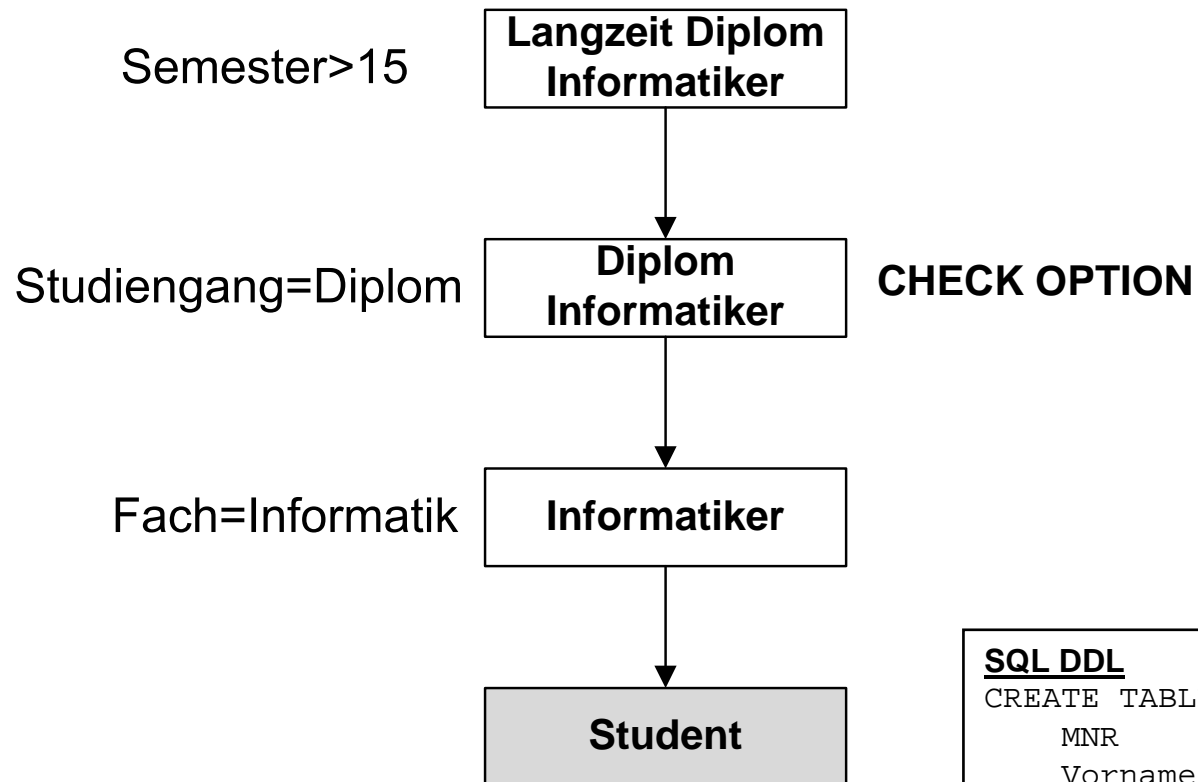
## **SQL DDL**

```
CREATE VIEW Informatiker AS  
    SELECT *  
    FROM Student  
    WHERE Fach='Informatik'
```

# Änderbarkeit in Sichten (1/2)

- Sichten gelten als NICHT änderbar, wenn:
  - der Primärschlüssel fehlt
  - eine Gruppierung oder/und Aggregation angewendet wird
  - mehrere Tabellen mit Join oder Kreuzprodukt verknüpft werden
- Um sicher zu stellen, dass geänderte Tupel nicht aus der Sicht verschwinden, werden Check Options (wir betrachten nur den Typ CASCADED Check Option) verwendet.
- Ist eine Sicht mit einer Check Option versehen, muss das geänderte Tupel
  - alle Bedingungen der betreffenden Sicht
  - alle Bedingungen der Sichten auf denen die betreffende Sicht aufbaut erfüllen damit die Änderungsoperation zulässig ist.

## Änderbarkeit in Sichten (2/2)



### SQL DDL

```
CREATE TABLE Student (  
    MNR          int          PRIMARY KEY,  
    Vorname      varchar(50),  
    Nachname     varchar(50) NOT NULL,  
    Fach         varchar(50) NOT NULL,  
    Studiengang  varchar(50) NOT NULL,  
    Semester     int          NOT NULL,  
);
```

# SQL-DML: Änderungen/Löschen/Einfügen

- Einfügen (Beispiel)

```
INSERT INTO Student (SID, Name, Wohnort, Fach, Geburtsdatum)
VALUES (47, 'Müller', 'Hamburg', 2, '1980-11-30');
```

- Ändern (Beispiel)

```
UPDATE Student
SET Fach = 3
WHERE SID = 47;
```

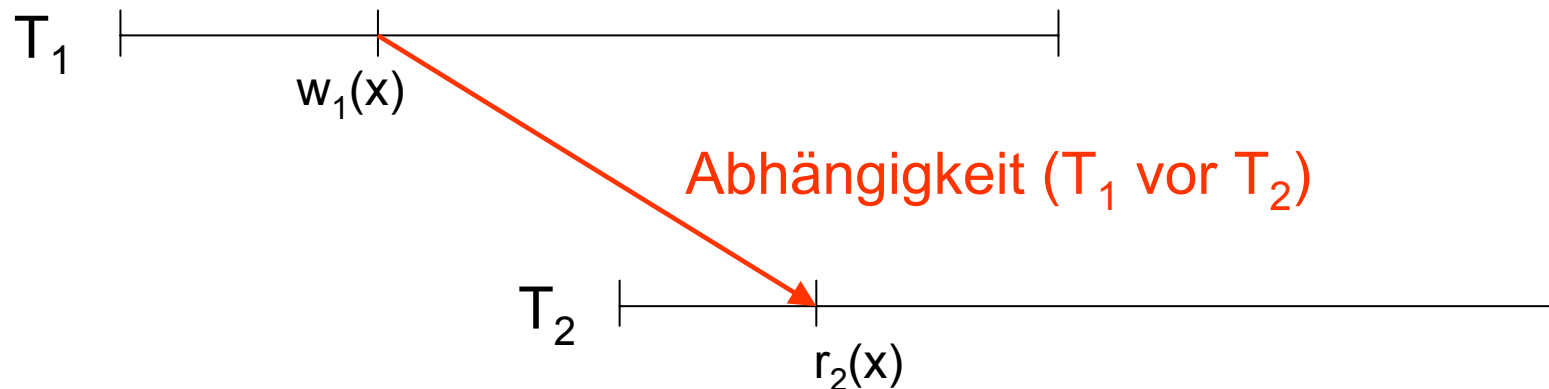
- Löschen (Beispiel)

```
DELETE FROM Student
WHERE SID = 47;
```



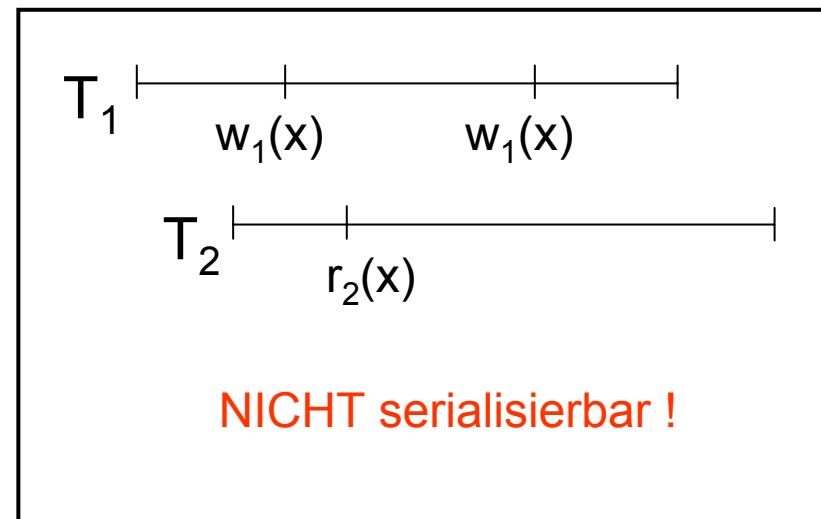
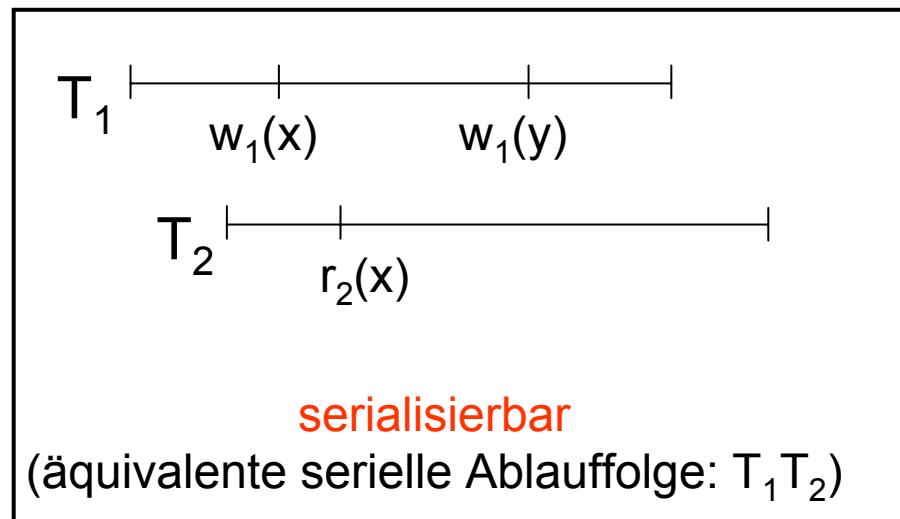
# Transaktionsverwaltung: Abhängigkeiten

- Zwei Transaktionen sind voneinander abhängig, wenn:
  - beide Transaktionen auf dasselbe Objekt zugreifen und
  - mindestens eine der Transaktionen auf dieses Objekt schreibt



# Transaktionsverwaltung: Serialisierbarkeit

- Eine parallele Ablauffolge, bestehend aus  $n$  Transaktionen ist serialisierbar, wenn:
    - eine serielle Ablauffolge dieser Transaktionen existiert, welche die gleichen Abhängigkeiten enthält
- => (keine Abhängigkeitszyklen existieren)



# RX-Sperrverfahren

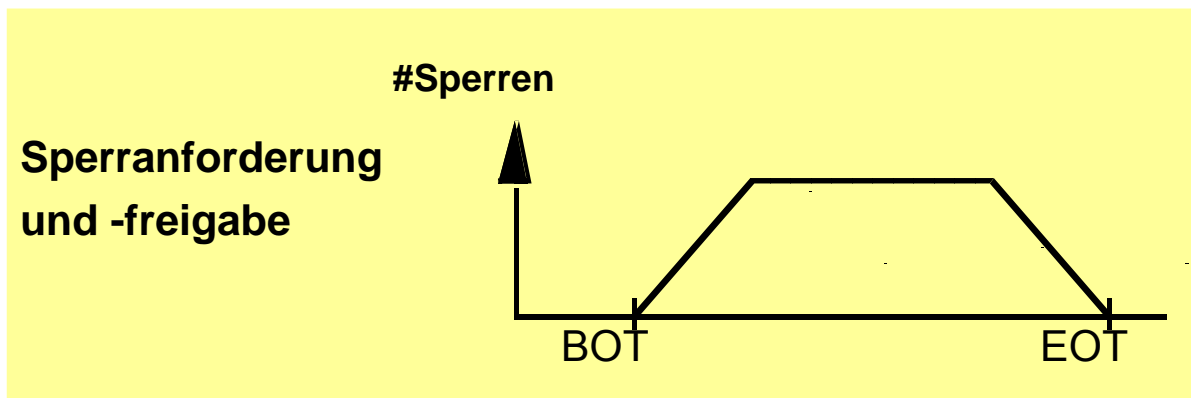
- Sperrmodi
  - Sperrmodus des Objektes: NL (no lock), R (read), X (exclusive)
  - Sperranforderung einer Transaktion: R, X
- Kompatibilitätsmatrix

	NL	R	X
R	+	+	-
X	+	-	-

- Falls Sperre nicht gewährt werden kann, muss die anfordernde TA warten, bis das Objekt freigegeben wird (Commit/Abort der die Sperre besitzenden TA)

# 2-Phasen-Sperrprotokoll (2PL)

- 2 Phasen
  - Lock-Phase (hier werden alle Sperren gesetzt)
  - Unlock-Phase (hier werden alle Sperren freigegeben)
- Prinzip: Es werden erst alle Sperren gesetzt bevor wieder eine Sperre freigegeben werden darf
- Vorteil: Gewährleistet einen serialisierbaren Schedule
- Nachteil: Kann verklemmen



# RX-Sperrverfahren mit 2PL

**$S_1 = w_1(a) \ r_2(c) \ r_3(a) \ r_1(b) \ c_1 \ w_2(c) \ r_3(b) \ c_2 \ c_3$**

Zeit	$T_1$	$T_2$	$T_3$	a	b	c	Bemerkungen
0				NL	NL	NL	
1	lock(a,X)			$X_1$	NL	NL	
2	write(a)	lock(c,R)		$X_1$	NL	$R_2$	
3		read(c)	lock(a,R)	$X_1$	NL	$R_2$	$T_3$ wartet auf Freigabe von a
4	lock(b,R)			$X_1$	$R_1$	$R_2$	
5	read(b)	lock(c,X)		$X_1$	$R_1$	$X_2$	
6	unlock(a)	write(c)		$R_3$	$R_1$	$X_2$	Benachritigung von $T_3$
7	unlock(b)	unlock(c)	read(a)	$R_3$	NL	NL	
8	commit	commit	lock(b,R)	$R_3$	$R_3$	NL	
...			...	...	...	...	

# RX-Sperrverfahren - Deadlocks

- Deadlocks/Verklemmungen
  - Auftreten von Verklemmungen ist **inhärent** und kann bei pessimistischen Methoden (blockierende Verfahren) nicht vermieden werden
- Beispiel eines nicht-serialisierbaren Schedules, der zu einer Verklemmung führt

