

# Kapitel 4: Transaktionen und ihre Realisierung

---

## Lernziele und Überblick:

- ❑ Transaktionen, Eigenschaften von Transaktionen
- ❑ Integritätsbedingungen
- ❑ Isolation von Transaktionen:
  - Terminologie und Formalisierung
  - Klassifikation der Probleme durch Verletzung der Isolation
  - Isolation durch Sperrprotokolle
- ❑ Diskussion: Vor- und Nachteile niedriger Isolationsgrade

# Integritätssicherung in SQL (1)

---

SQL erzwingt die folgenden *SQL-inhärenten Integritätsbedingungen* durch textuelle Analyse der Anweisungen unter Benutzung der Schemainformationen (□ *statische Typisierung* in Programmiersprachen):

- **Typisierung der Spalten:** In einer Spalte können nur typkompatible Werte gespeichert werden.
- **Homogenität der Reihen:** Alle Reihen einer Tabelle besitzen eine identische Spaltenstruktur.

Zur *applikationsspezifischen Integritätssicherung* stehen zwei syntaktische Konstrukte zur Verfügung, die beide Boole'sche Prädikate zur *deklarativen* Integritätssicherung benutzen und zur Laufzeit erzwungen werden:

- **Domänenzusicherungen:**

Basistypen mit zugehörigen Zusicherungen können in Form benannter *SQL-Domänen* im aktuellen Schema definiert werden:

```
create domain Schulnote integer
constraint NoteDefiniert check(value is not null)
constraint NoteZwischen1und6 check(value in(1,2,3,4,5,6));
```

# Integritätssicherung in SQL (2)

---

- ❑ **Tabellenzusicherungen** werden syntaktisch in Tabellendefinitionen geschachtelt. Sie garantieren, daß die Auswertung des Prädikats in jedem Datenbankzustand den Wert *true* liefert (universelle Quantifizierung).

```
create table Tabellename (...  
    constraint Zusicherungsname  
    check (Prädikat))
```

- ❑ **Schemazusicherungen** sind SQL-Objekte, die dynamisch dem aktuellen SQL-Schema hinzugefügt werden können. Sie garantiert, daß in jedem Datenbankzustand die Auswertung des Prädikats den Wert *true* liefert.

```
create assertion Zusicherungsname  
    check (Prädikat) ;
```

Ein Datenbankzustand heißt konsistent, wenn er alle im Schema deklarierten Zusicherungen erfüllt. Logisch gesehen sind alle Tabellen- und Schemazusicherungen konjunktiv verknüpft.

# Spaltenwertintegrität

---

Eine Tabellenzusicherung, deren Prädikat sich nur auf einen Spaltennamen bezieht, garantiert die Spaltenintegrität und wird in folgenden Modellierungssituationen eingesetzt:

- ❑ Vermeidung von Nullwerten
- ❑ Definition von Unterbereichstypen
- ❑ Definition von Formatinformationen durch Stringvergleiche
- ❑ Definition von Aufzählungstypen

```
check(Alter is not null)
```

```
check(Alter >=0 and Alter <=150)
```

```
check(Postleitzahl like 'D-____')
```

```
check(Note in (1,2,3,4,5,6))
```

# Reihenintegrität

---

Eine Tabellenzusicherung, deren Prädikat sich auf mehrere Spaltennamen bezieht, definiert eine Reihenintegritätsbeziehung, die von jeder Reihe einer Tabelle erfüllt sein muß.

```
check (Ausgaben <= Einnahmen)

check ((HatVordiplom, HatDiplom) in values (
  ('nein', 'nein')
  ('ja', 'nein')
  ('ja', 'ja')))

```

# Tabellenintegrität (1)

---

Die Überprüfung quantifizierter Prädikate kann im Gegensatz zu den bisher besprochenen Zusicherungen im schlimmsten Fall die Auswertung einer kompletten mengenorientierten Anfrage zur Folge haben:

```
check((select sum(Budget) from Projekte) >= 0)

check(exists(select * from Abteilung
              where Oberabt = 'LTSW'))
```

Einige in der Praxis häufig vorkommenden quantifizierte Zusicherungen können mittels *Indexstrukturen* (B-Bäume, Hash-Tabelle, s. Kapitel 9) effizient überprüft werden und sogar zu einem *Effizienzgewinn* bei Anfragen und Änderungsoperationen führen.

# Tabellenintegrität (2)

---

Für häufig auftretende Muster von quantifizierten Zusicherungen bietet SQL syntaktische Konstrukte an, was die *Lesbarkeit* erhöht und *optimierende Implementierung* ermöglicht:

- 1. Spaltenwerteindeutigkeit:** Die Eindeutigkeit von Spaltenwertkombinationen in einer Tabelle gestattet eine wertbasierte Identifikation von Tabellenelementen (□ *Schlüsselkandidat*).

Beispiel zweier semantisch äquivalenter Zusicherungen:

```
create table Projekte(...  
  unique (Name) )
```

```
create table Projekte(...  
  check (all x, all y: ...  
    (  
      (x.Name <> y.Name or x = y)  
    )  
  )
```

$(x.Name = y.Name) \rightarrow (x = y)$

Eine Tabelle kann mehrere Schlüsselkandidaten besitzen, die durch separate unique-Klauseln beschrieben werden.

# Tabellenintegrität (3)

---

**2. Primärschlüsselintegrität:** Ein Schlüsselkandidat, in dessen Spalten keine Nullwerte auftreten dürfen, kann als Primärschlüssel ausgezeichnet werden. Eine Tabelle kann nur einen Primärschlüssel besitzen.

Beispiel zweier semantisch äquivalenter Zusicherungen:

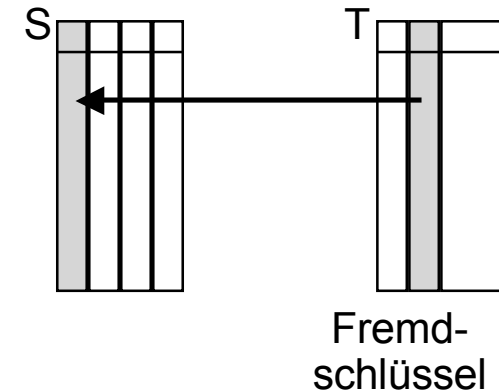
```
create table Projekte (...  
    primary key(Nr) )
```

```
create table Projekte(...  
    unique Nr  
    check(Nr is not null), )
```

**3. Referentielle Integrität (Fremdschlüsselintegrität):** Diese Zusicherung bezieht sich auf den Zustand zweier Tabellen (s. nächste Folien)

# Referentielle Integrität (1)

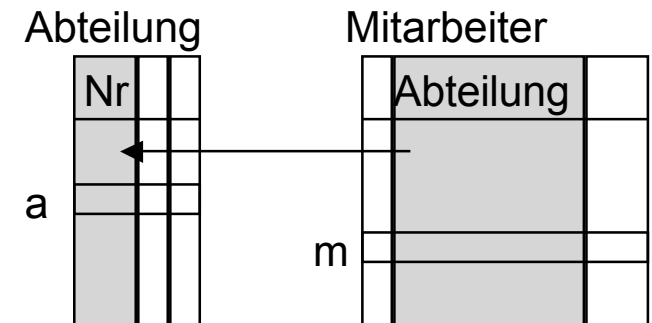
Referentielle Integrität ist eine Zusicherung über den Zustand zweier Tabellen, die dann erfüllt ist, wenn zu jeder Reihe in Tabelle *T* eine zugehörige Reihe in Tabelle *S* existiert, die den Fremdschlüsselwert von *T* als Wert ihres Schlüsselkandidaten besitzt.



Beispiel zweier semantisch äquivalenter Zusicherungen:

```
create table Mitarbeiter
(...
  constraint MitarbeiterHatAbteilung
  foreign key (Abteilung)
  references Abteilung (Nr))
```

```
create constraint MitarbeiterHatAbteilung
  check (not exists (select * from Mitarbeiter m where
    not exists (select * from Abteilung a where m.Abteilung = a.Nr)))
```



□ m □ Mitarbeiter : □ a □ Abteilung: m.Abteilung=a.Nr

# Referentielle Integrität (2)

---

Im allgemeinen besteht ein Fremdschlüssel einer Tabelle  $T$  aus einer Liste von Spalten, der eine typkompatible Liste von Spalten in  $S$  entspricht:

```
create table T
  (...
  constraint Name
    foreign key (A1, A2, ..., An) references (S (B1, B2, ..., Bn))
```

Sind  $B_1, B_2, \dots, B_n$  die Primärschlüsselspalten von  $T$ , kann ihre Angabe entfallen.

**Beachte:** Rekursive Beziehungen (z.B. Abteilung : Oberabteilung) führen zu reflexiven Fremdschlüsseldeklarationen ( $S = T$ ).

# Behandlung von Integritätsverletzungen (1)

---

- ❑ Ohne spezielle Maßnahmen wird eine SQL-Anweisung, die eine Zusicherung verletzt, vom DBMS ignoriert. Eine Statusvariable signalisiert, welche Zusicherung verletzt wurde.
- ❑ Bei der commit-Anweisung wird im Falle einer gescheiterten verzögerten Integritätsbedingung ein Transaktionsabbruch (`rollback`) ausgelöst. Es kann daher sinnvoll sein, vor dem Transaktionsende mit der Anweisung `set constraints all immediate` eine unmittelbare Überprüfung aller verzögerbaren Zusicherungen zu erzwingen und Integritätsbedingungen explizit programmgesteuert zu behandeln.
- ❑ Fremdschlüsselintegrität zwischen zwei Tabellen *S* und *T* kann durch vier Operationen verletzt werden:
  1. `insert into T`
  2. `update T set ...`
  3. `delete from S`
  4. `update S set ...`

# Behandlung von Integritätsverletzungen (2)

---

- ❑ Im Fall 1 und 2 führt der Versuch in  $T$  einen Fremdschlüsselwert einzufügen, der nicht in  $S$  definiert ist dazu, daß die Anweisung ignoriert wird. Die Verletzung wird über eine Statusvariable oder eine Fehlermeldung angezeigt.
- ❑ Wird im Falle 3 oder 4 versucht, eine Reihe zu löschen, deren Schlüsselwert noch als Fremdschlüsselwert in einer oder mehrerer Reihen der Tabelle  $T$  auftritt, wird eine der folgenden Aktion ausgeführt, die am Ende der references-Klausel spezifiziert werden kann; folgende Aktionsalternativen sind möglich:
  - **set null**: Der Fremdschlüsselwert aller betroffener Reihen von  $T$  wird durch **null** ersetzt.
  - **set default**: Der Fremdschlüsselwert aller betroffener Reihen von  $T$  wird durch den Standardwert der Fremdschlüsselspalte ersetzt.
  - **cascade**: Im Fall 3 (**delete**) werden alle betroffenen Reihen von  $T$  gelöscht. Im Falle 4 (**update**) werden die Fremdschlüsselwerte aller betroffenen Reihen von  $T$  durch die neuen Schlüsselwerte der korrespondierenden Reihen ersetzt.
  - **no action**: Es wird keine Folgeaktion ausgelöst, die Anweisung wird ignoriert.

# Zeitpunkt der Integritätsprüfung

---

Bezieht sich eine Zusicherung auf mehrere Zustandsvariablen, muß der Zeitpunkt der Integritätsprüfung nach Änderungsoperationen genau spezifiziert werden.

Dazu existieren zwei Modi der Integritätsprüfung:

- ❑ **not deferrable** kennzeichnet eine nicht verzögerbare Zusicherung, die unmittelbar nach jeder SQL-Anweisung überprüft wird.
- ❑ **deferrable** kennzeichnet eine verzögerbare Zusicherung. Man kann ein Flag auf den Wert
  - **immediate** setzen, wenn nach der nächsten SQL-Anweisung geprüft werden soll oder auf den Wert
  - **deferred**, wenn die Prüfung bis zum Transaktionsende aufgeschoben werden soll.
  - Zusätzlich wird bei jedem Umschalten auf den Wert **immediate** und am Transaktionsende überprüft.

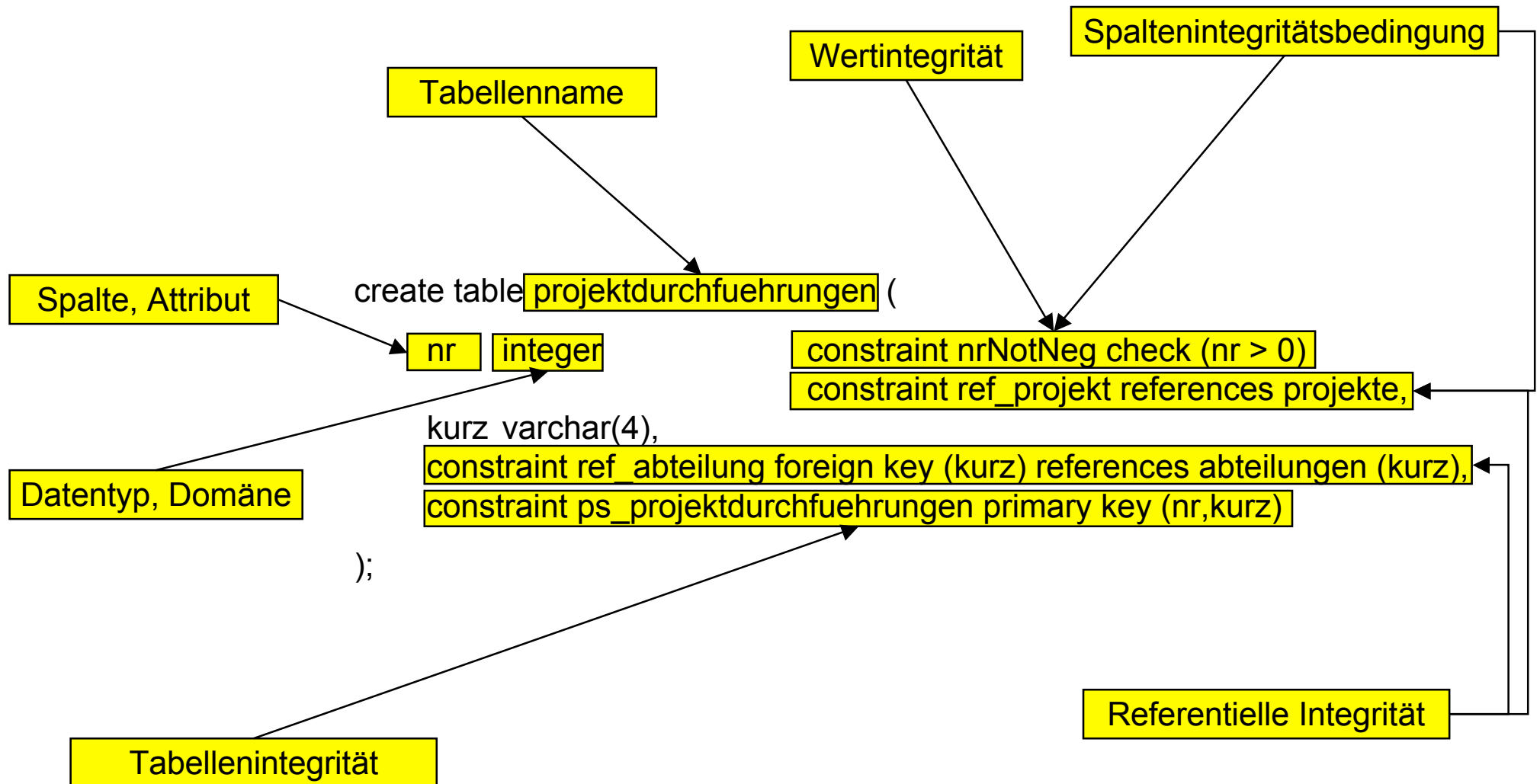
Optimierungen können den tatsächlichen Zeitpunkt beeinflussen.

# Zusammenfassung: Tabellendefinition

---

```
create table projektdurchfuehrungen (  
    nr    integer                constraint nrNotNeg check (nr > 0)  
                                constraint ref_projekt references projekte,  
    kurz varchar(4),  
    constraint ref_abteilung foreign key (kurz) references abteilungen (kurz),  
    constraint ps_projektdurchfuehrungen primary key (nr,kurz)  
);
```

# Zusammenfassung: Tabellendefinition



# Zusammenfassung: Anfragen

---

```
select a.name, p.titel as Projekt
from abteilungen a, projekte p, projektdurchfuehrungen pd
where a.kurz = pd.kurz and p.nr = pd.nr
order by a.name asc, Projekt asc
```

# Zusammenfassung: Anfragen

