Datenbanken: SQL

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Dennis Heinrich (Übungen)

RDM: Projektdatenbank

Nr	*Titel*	*Budget*
100	DB Fahrpläne	300.000
200	ADAC Kundenstamm	100.000
300	Telekom Statistik	200.000

Projekte

Nr	*Kurz*
100	MFSW
100	UXSW
100	LTSW
200	UXSW
200	PERS
300	MFSW

Projektdurchführung

Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
UXSW	Unix SW	LTSW
PCSW	PC SW	LTSW
LTSW	Leitung SW	**NULL**
PERS	Personal	**NULL**

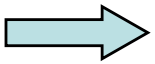
Abteilungen

Projektdatenbank

SQL: Einfache Anfragen (ohne Variable)

Projektion und Selektion: SQL-Anfrage zur Bestimmung der Namen und des Kürzels aller Abteilungen, die der Abteilung 'Leitung Software' mit dem Kürzel *LTSW* untergeordnet sind

```
select Name, Kurz
from Abteilungen
where Oberabt = 'LTSW';
```

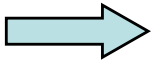


Ergebnistabelle

Name	*Kurz*
Mainframe SW	MFSW
Unix SW	UXSW
PC SW	PCSW

Selektion (ohne Projektion): Aufzählung *aller* Spalten (durch * in der *Projektionsliste*) der Bereichstabelle unter Beibehaltung der Spaltenreihenfolge

```
select *
from Abteilungen
where Oberabt
        = 'LTSW';
```



Ergebnistabelle

Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
UXSW	Unix SW	LTSW
PCSW	PC SW	LTSW

SQL: Komplexere Anfrage (mit Variablen)

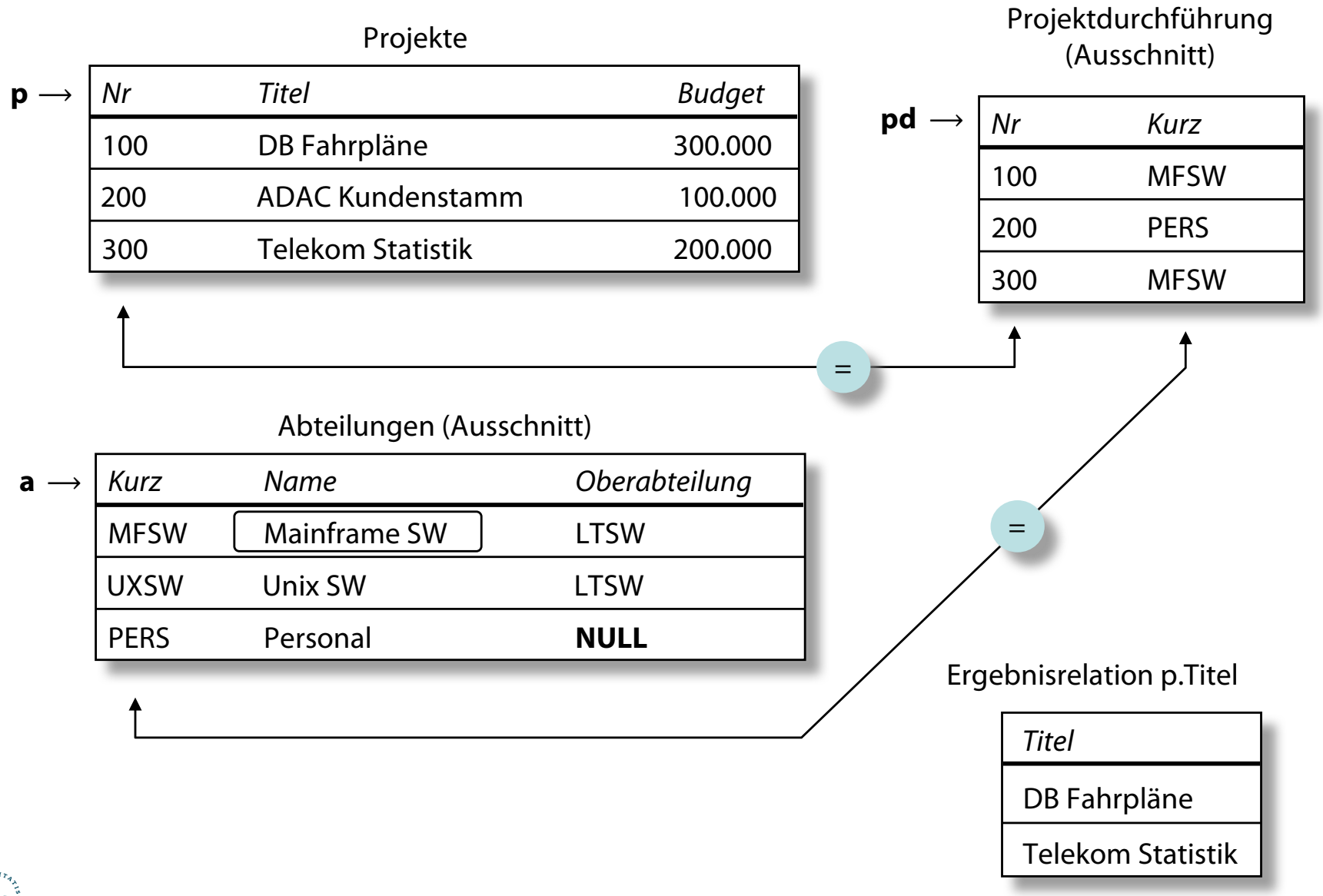
Iterationsabstraktion mit Hilfe des **select from where**-Konstrukts:

- **select**-Klausel: Spezifikation der Projektionsliste für die Ergebnistabelle
- **from**-Klausel: Festlegung der angefragten Tabellen, Definition und Bindung der Tupelvariablen
- **where**-Klausel: Selektionsprädikat, mit dessen Hilfe die Ergebnistupel aus dem kartesischen Produkt der beteiligten Tabellen selektiert werden

Bestimmung der Projekttitel, an denen die Abteilung für *Mainframe Software* arbeitet:

```
select p.Titel
from Projekte p,
     Projektdurchfuehrung pd,
     Abteilungen a
where p.Nr = pd.Nr
and a.Kurz = pd.Kurz
and a.Name = 'Mainframe SW';
```

Join im Where-Teil



Vermeidung von Variablen

```
select Titel
from Projekte
     natural join
     Projektdurchfuehrung
     natural join
     Abteilungen
where Name = 'Mainframe SW';
```

Join-Operatoren:

- <table> CROSS JOIN <table>
- <table> NATURAL JOIN <table>
- <table> [INNER] JOIN <table> [ON <cond>]
- <table> (LEFT | RIGHT | FULL) [OUTER] JOIN <table> [ON <cond>]

RDM: Aktualisierungsoperationen

Änderungsoperationen beziehen sich auf Relationen oder Teilrelationen (select …):

- **insert-Statement:**
  - Fügt ein einziges Tupel ein, dessen Attributwerte als Parameter übergeben werden.
  - Fügt eine Ergebnistabelle ein.
- **update-Statement:**
  - Selektion (des) der betreffenden Tupel(s)
  - Neue Werte oder Formeln für zu ändernde Attribute
- **delete-Statement:**
  - Selektion (des) der betreffenden Tupel(s)

```
insert into Projektdurchfuehrung
values (400, 'XYZA')
```

```
insert into Projektdurchfuehrung
   (Nr, Kurz)
select p.Nr, a.Kurz
from Projekte p, Abteilungen a
where p.Titel = 'Telekom Statistik'
      and a.Name = 'Unix SW'
```

```
update Projekte
set Budget = Budget * 1.5
where Budget > 150000
```

```
delete
from Projektdurchfuehrung
where Kurz = 'MFSW';
```

Ausdrücke in der Projektionsliste

- Beispiel:

```
select Budget + 100000
from Projekte
where Budget > 200000;
```

- Auch selbstdefinierte Funktionen verwendbar (hier nicht näher behandelt)

Lexikalische und syntaktische Regeln (1)

Große Anzahl optionaler Klauseln und schlüsselwortbasierter Operatoren

SQL-Quelltext von Syntaxanalyse in Folge von Symbolen zerlegt

- Nicht-druckbare Steuerzeichen (z.B. Zeilenvorschub) und Kommentare wie Leerzeichen behandelt
- Kommentare beginnen mit --
  und reichen bis zum Zeilenende
- Kleinbuchstaben in Großbuchstaben umgewandelt,
  falls sie nicht in Zeichenketten-Konstanten auftreten

Lexikalische und syntaktische Regeln (2)

- **Reguläre Namen** beginnen mit einem Buchstaben gefolgt von evtl. weiteren Buchstaben, Ziffern und _
- **Schlüsselworte**: SQL definiert über 210 Namen als Schlüsselworte, die nicht kontextsensitiv sind
- **Begrenzte Namen**: Zeichenketten in doppelten Anführungszeichen (Verwendung von Schlüsselworten als Namen)
- **Literale** dienen zur Benennung von Werten der SQL-Basistypen
- weitere Symbole (Operatoren etc.)

```
Peter, mary33
```

```
create, select
```

```
"intersect", "create"
```

```
'abc'       character(3)
123         smallint
B'101010'   bit(6)
```

```
<, >, =, %, &, (, ),
*, +, ...
```

Schemata und Kataloge (1)

- SQL-Schema ist *dynamischer Sichtbarkeitsbereich* für Namen geschachtelter (lokaler) SQL-Objekte (Tabellen, Sichten, Regeln ...)

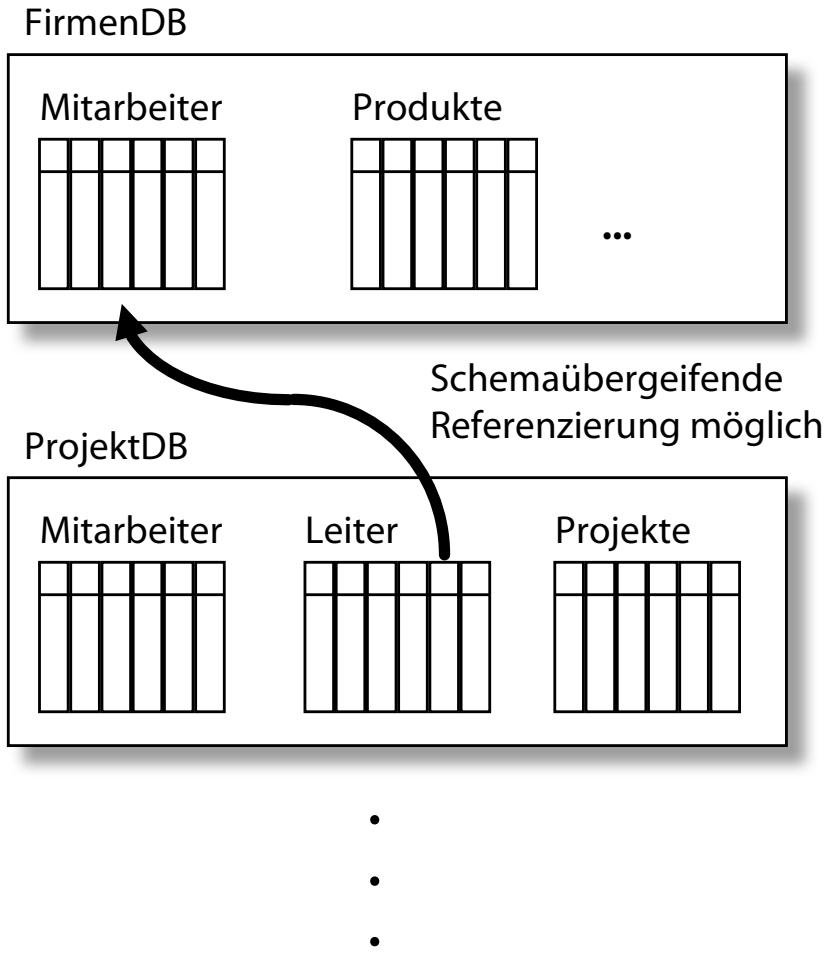
```
create schema FirmenDB;
  create table Mitarbeiter ...;
  create table Produkte ... ;

create schema ProjektDB;
  create table Mitarbeiter ...;
  create view Leiter ...;
  create table Projekte ...;
  create table Test ...;
  drop table Test;

drop schema FirmenDB;
```

- Schemata werden persistent gespeichert (zugreifbar über SQL)
- Multiple Schemata nötig für:
  - Integration separat entwickelter Datenbanken
  - Arbeit in verteilten und föderativen Datenbanken

Schemata und Kataloge (2)

Schemakatalog

Name	Benutzer	
FirmenDB	matthes	
ProjektDB	matthes	
TextDB	schmidt	




```
FirmenDB.Mitarbeiter
ProjektDB.Mitarbeiter
```

```
create schema FirmenDB
connect FirmenDB
```

Schemata und Kataloge (3)

- *Schemaabhängigkeiten* entstehen durch Referenzen von SQL-Objekten eines Schemas in ein anderes Schema.

```
create view ProjektDB.Leiter as
  select * from FirmenDB.Mitarbeiter
  where ...
```

- Schemaabhängigkeiten müssen beim Löschen eines Schemas berücksichtigt werden. **cascade** erzwingt das transitive Löschen der abhängigen SQL-Objekte

```
drop schema FirmenDB cascade
```

- Anlegen und Löschen eines SQL-Schemas impliziert Anlegen bzw. Löschen der Datenbank, die das Schema implementiert

- Schemata sind wiederum in Sichtbarkeitsbereichen enthalten, den **Katalogen** (Kataloge können geschachtelt werden)
  - Kataloge enthalten weitere Information wie z.B. Zugriffsrechte, Speichermedium, Datum des letzten Backup, ...

Basisdatentypen und Typkompatibilität (1)

- Formale Definition des relationalen Datenmodells basiert auf einer Menge von Domänen, der die atomaren Werte der Attribute entstammen

- Anforderungen an die algebraische Struktur einer Domäne *D*:

  - Existenz einer Äquivalenzrelation auf *D* zur Definition der Relationensemantik (⟶ *Duplikatelimination*) und des Begriffs der funktionalen Abhängigkeit

  - Existenz weiterer Boolescher Prädikate (>, <, >=, substring, odd, ...) auf *D* zur Formulierung von Selektions- und Joinausdrücken über Attribute

  - Moderne erweiterbare Datenbankmodelle unterstützen auch benutzerdefinierte Domänen

Basisdatentypen und Typkompatibilität (2)

SQL hält den Datenbankzustand und die Semantik von Anfragen unabhängig von speziellen Programmen und Hardwareumgebungen.

Festes Repertoire an anwendungsorientierten *vordefinierten Basisdatentypen*

- **Lexikalische Regeln** für Literale
- **Evaluationsregeln** für unäre, binäre und n-äre Operatoren (Wertebereich, Ausnahmebehandlung, Behandlung von Nullwerten)
- **Typkompatibilitätsregeln** für gemischte Ausdrücke
- **Wertkonvertierungsregeln** für den bidirektionalen Datenaustausch mit typisierten Programmiersprachenvariablen bei der Gastspracheneinbettung.
- Spezifikation des **Speicherbedarfs** (minimal, maximal) für Werte eines Typs.

SQL bietet zahlreiche standardisierte Operatoren auf Basisdatentypen und erhöht damit die Portabilität der Programme.

Basisdatentypen und Typkompatibilität (3)

- **Exact numerics** bieten exakte Arithmetik und gestatten die Angabe einer Gesamtlänge und der Nachkommastellenzahl.
- **Approximate numerics** bieten aufgrund ihrer Fließkommadarstellung einen flexiblen Wertebereich, sind jedoch wegen der Rundungsproblematik nicht für kaufmännische Anwendungen geeignet.
- **Character strings** beschreiben mit Leerzeichen aufgefüllte Zeichenketten fester Länge oder variabel lange Zeichenketten mit fester Maximallänge (auch: **char** oder **varchar**)
- **Bit strings** beschreiben mit Null aufgefüllte Bitmuster fester Länge oder variabel lange Bitfelder mit fester Maximallänge.

```
integer, smallint,
numeric(p,s),
decimal(p,s)
```

```
real,
double precision,
float(p)
```

```
character(n),
character varying(n)
```

```
bit(n),
bit varying(n)
```

Basisdatentypen und Typkompatibilität (4)

- **Datetime** Basistypen beschreiben Zeit(punkt)werte vorgegebener Granularität.
- **Time intervals** beschreiben Zeitintervalle vorgegebener Dimension und Granularität.

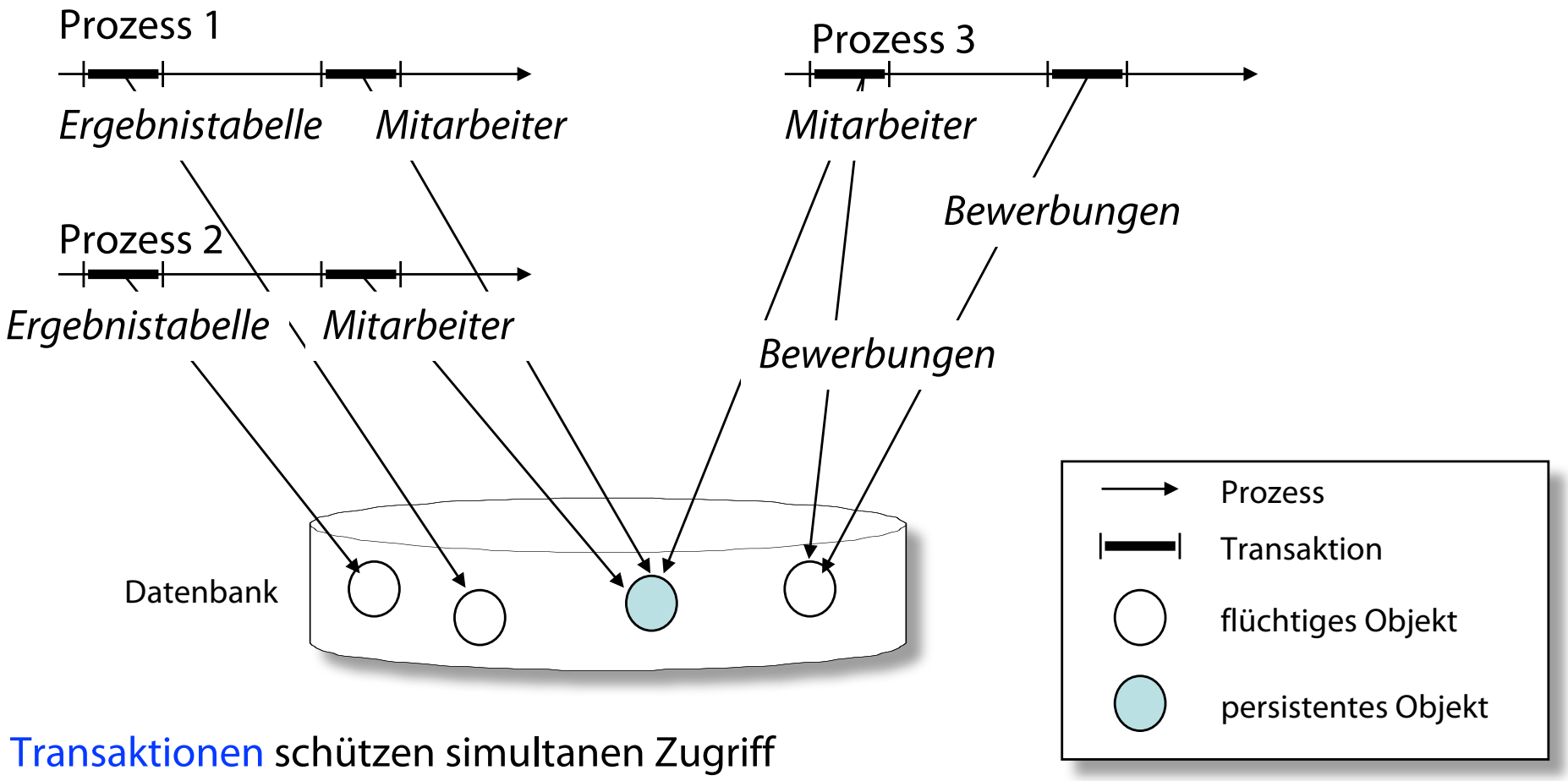
```
date, time(p), timestamp,
time(p) with time zone,
```

```
interval year(2) to month
```

SQL unterstützt sowohl die implizite Typanpassung (*coercion*), als auch die explizite Typanpassung (*casting*).

Lebensdauer, Sichtbarkeit, gemeinsame Nutzung (1)

Die gleiche Datenbank kann von verschiedenen informationsverarbeitenden Prozessen simultan oder sequentiell nacheinander benutzt werden.



Transaktionen schützen simultanen Zugriff

Standardwerte für Spalten

Beim Einfügen von Reihen in eine Tabelle können einzelne Spalten unspezifiziert bleiben.

```
insert into Mitarbeiter
  (Name, Gehalt, Urlaub)
values ('Peter', 3000, null)
```

```
insert into Mitarbeiter
  (Name, Gehalt)
values ('Peter', 3000)
```

Fehlende Werte werden mit **null** oder mit bei der Tabellenerzeugung angegebenen Standardwerten belegt.

- Standardwerte können Literale eines Basisdatentyps sein.
- Standardwerte können eine parameterlose SQL-Funktion sein, die zum Einfügezeitpunkt ausgewertet wird.

Datenunabhängigkeit und *Schemaevolution*:

- Existierende Anwendungsprogramme können auch nach dem Erweitern einer Relation konsistent mit neu erstellten Anwendungen interagieren

Null

- Jeder SQL-Basisdatentyp ist zur Unterstützung solcher Modellierungssituationen um den ausgezeichneten Wert **null** erweitert, der von jedem anderen Wert dieses Typs verschieden ist (auch: NULL ≠ NULL)
- Null ist Default-Wert sofern möglich und nicht speziell definiert
- Das Auftreten von Nullwerten in Attributen oder Variablen kann verboten werden

```
integer not null
```

```
CREATE TABLE Persons (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255) DEFAULT 'Sandnes');
```

Annahmen

- Close-World Assumption (CWA)
- "Das mit der DB beschriebene Modell ist vollständig"

Beispiel

Tabelle Uni-Angestellter

ID	Name
1	Sokrates
2	Platon
3	Aristoteles

Tabelle Professor

ID
1
2

"3" (= ID von Aristoteles) kommt nicht in Tabelle Professor vor
$\Longrightarrow$ Aristoteles ist kein Professor

Unvollständigkeit in den Daten

- Close-World Assumption (CWA)
- "Das mit der DB beschriebene Modell ist vollständig"

Beispiel

Tabelle Patient

ID	Name
1	Sokrates
2	Platon
3	Aristoteles

Tabelle Patient-Blutzucker

ID	Blutzuckerwert
1	90
2	120

"3" kommt nicht in Tabelle Patient-Blutzuckerwert vor

$\Longrightarrow$? Aristoteles hat keinen Blutzuckerwert.

NULL für nicht bekannt

- NULLs für Modellierung von Unvollständigkeit
- Aber Semantik nicht geklärt und daher häufig kritisiert

L. Libkin. SQL's three-valued logic and certain answers. ACM Trans. Database Syst., 41(1):1:1–1:28, 2016. (s. auch Vorl. Foundations of Databases and Ontologies)

Beispiel

Tabelle Patient

ID	Name
1	Sokrates
2	Platon
3	Aristoteles

Tabelle Blutzucker

ID	Blutzuckerwert [30-600]
1	90
2	120
3	NULL

Aristoteles hat einen Blutzuckerwert (30 oder 31 oder ...)

Nicht bekannt oder nicht anwendbar?

- NULLs für Modellierung von Unvollständigkeit
- Aber Semantik nicht geklärt und daher häufig kritisiert

L. Libkin. SQL's three-valued logic and certain answers. ACM Trans. Database Syst., 41(1):1:1–1:28, 2016. (s. auch Vorl. Foundations of Databases and Ontologies)

Beispiel

Tabelle Patient

ID	Name
1	Sokrates
2	Platon
3	Aristoteles
4	Xanthippe
5	Leda

Tabelle Schwangerschaft

ID	HCG-Hormon-Wert
1	NULL
2	NULL
3	NULL
4	NULL
5	130

- Männliche Patienten NULL: kein HCG-Test
- Weibliche Patienten mit NULL: kein HCG-Test (aber HCG-Wert hat sie) oder HCG-Test & nicht bekannt

Null-Werte

- Jeder SQL-Basisdatentyp um den ausgezeichneten Wert **null** erweitert (verschieden von jedem anderen Wert)
  - NULL ≠ NULL (z.B. beim Verbund)
- Null ist Default-Wert sofern möglich bzw. nicht speziell definiert
- Das Auftreten von Nullwerten in Attributen oder Variablen kann verboten werden (dann typspezifischer Default-Wert)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes');
```

Nullwerte und Wahrheitswerte

Wahrheitstabellen der dreiwertigen SQL-Logik:

OR	true	false	null
true	*true*	*true*	*true*
false	*true*	*false*	*null*
null	*true*	*null*	*null*

AND	true	false	null
true	*true*	*false*	*null*
false	*false*	*false*	*false*
null	*null*	*false*	*null*

x	not x	x is null	x is not null
true	*false*	*false*	*true*
false	*true*	*false*	*true*
null	*null*	*true*	*false*

Schwierigkeiten bei der konsistenten Erweiterung einer Domäne um Nullwerte werden bereits am einfachen Beispiel der Booleschen Werte und der grundlegenden logischen Äquivalenz **x and not x = false** deutlich, die bei der Erweiterung der Domäne um Nullwerte verletzt wird (**null and not null = null**)

Nullwerte und Wahrheitswerte

Vorteile:

- Explizite und konsistente Behandlung von Nullwerten durch alle Applikationen
  - Im Gegensatz zu ad hoc Lösungen, bei denen z.B. der Wert -1, *-MaxInt* oder die leere Zeichenkette als Null-Wert eingesetzt wird
- Definition der Semantik von Datenbankoperatoren bzgl. Null-Werten (Vergleich, Arithmetik)

Nachteile:

- *Konflikt* mit den algebraischen Eigenschaften (Existenz von Nullelementen, Assoziativität, Kommutativität, Ordnung, ...)
  - (... -2 < -1 < 0 < **Null** < 1 < 2 < ... ?)
- Null-Werte *verhindern* häufig *Anfrageoptimierung*
- Semantik trotzdem anwendungsabhängig (unbekannter Wert, n/a, ...)

RDM: Projektdatenbank

Nr	*Titel*	*Budget*
100	DB Fahrpläne	300.000
200	ADAC Kundenstamm	100.000
300	Telekom Statistik	200.000

Projekte

Nr	*Kurz*
100	MFSW
100	UXSW
100	LTSW
200	UXSW
200	PERS
300	MFSW

Projektdurchführung

Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
UXSW	Unix SW	LTSW
PCSW	PC SW	LTSW
LTSW	Leitung SW	**NULL**
PERS	Personal	**NULL**

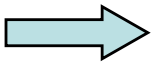
Abteilungen

Projektdatenbank

Duplikatelimination

Elimination von Duplikaten im Anfrageergebnis mit dem Schlüsselwort distinct:

```
select distinct Oberabt
from Abteilungen;
```



Oberabt
LTSW
NULL

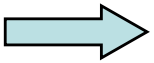
Hier: Umwandlung einer Ergebnistabelle in *Ergebnismenge*

Man beachte die Behandlung von Null-Werten

… und wenn null für verschiedene Werte steht?

Erkennung und Vermeidung von Nullwerten in Spalten durch das Prädikat **is null** oder **is not null**

```
select distinct Oberabt
from Abteilungen
where Oberabt is not null;
```

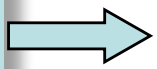


Oberabt
LTSW

Sortierordnung

Sortierte Darstellung der Anfrageergebnisse über die order by-Klausel mit den Optionen asc (*ascending, aufsteigend*) und desc (descending, absteigend):

```
select *
from Abteilungen
where Oberabt = 'LTSW'
order by Kurz asc;
```



Ergebnistabelle

Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
PCSW	PC SW	LTSW
UXSW	Unix SW	LTSW

Finden Sie heraus, was bei Null-Werten passiert bzw. wie man damit umgeht.

Die Sortierung kann mehrere Spalten umfassen:

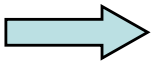
- Aufsteigende Sortierung aller Abteilungen gemäß Namen der Oberabteilung
- Anschließend für gleiche Oberabteilungen Sortierung absteigend nach Kurz

```
select *
from Abteilungen
order by Oberabt asc,
         Kurz desc;
```

Aggregatfunktionen

- Nutzung in der select-Klausel einer SQL-Anwendung
- Berechnung aggregierter Werte
  (z.B. Summe über alle Werte einer Spalte einer Tabelle)
- **Beispiel**: Summe und Maximum der Budgets aller Projekte

```
select sum(p.Budget),
       max(p.Budget)
from Projekte p;
```

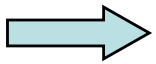


p.Budget

sum	*max*
600.000	300.000

- Auch: Minimum (**min**), Durchschnitt (**avg**),
  Zählen der Tabellenwerte einer Spalte (**count**)
  bzw. der Anzahl der Tupel (**count(*)**)
- **Beispiel**: Anzahl der Tupel in der Relation Abteilungen (inkl. Nullwerte und Duplikate)

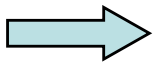
```
select count(*)
from Abteilungen;
```



count(*)
5

Einmaliges Zählen
von Werten möglich
(nur Nicht-Nullwerte)

```
select
  count(distinct Oberabt)
from Abteilungen;
```

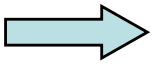


count(*)
1

Ausdrücke in der Projektionsliste

```
select Budget + 100000
from Projekte
where Budget > 200000;
```

```
select sum(p.Budget) + 100000,
       max(p.Budget)
from Projekte p;
```



p.Budget

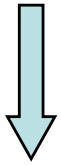
sum	*max*
700.000	300.000

```
select Name || 'Temp', Kurz
from Abteilungen
where Oberabt = 'LTSW';
```

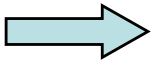
Gruppierung: Beispiel

Gib zu jeder Oberabteilung die Anzahl der Unterabteilungen an

```
select Oberabt, count(Kurz)
from Abteilungen
group by Oberabt;
```



Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
UXSW	Unix SW	LTSW
PCSW	PC SW	LTSW
LTSW	Leitung SW	**NULL**
PERS	Personal	**NULL**



Ergebnistabelle

Oberabt	*count(Kurz)*
LTSW	3
NULL	2

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten

PersINr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Aggregatfunktion und Gruppierung

Aggregatfunktionen **avg, max, min, count, sum**

```
select avg(Semester)
from Studenten;
```

```
select gelesenVon, sum(SWS)
from Vorlesungen
group by gelesenVon;
```

```
select gelesenVon, Name, sum(SWS)
from Vorlesungen, Professoren
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon, Name
having avg (SWS) >= 3;
```

Attribut Name muss vorkommen

Ausführen einer Anfrage mit group by

Vorlesung x Professoren							
VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **where**-Bedingung

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Gruppierung

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **having**-Bedingung

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Aggregation (**sum**) und Projektion

Ergebnis

gelesenVon	Name	**sum** (SWS)
2125	Sokrates	10
2137	Kant	8

Gruppierung

- Zusammenfassung von Zeilen einer Tabelle in Abhängigkeit von Werten in bestimmten Spalten, den *Gruppierungsspalten*
  - Alle Zeilen einer Gruppe enthalten in dieser Spalte bzw. diesen Spalten den gleichen Wert
  - Pro Gruppe ein Ergebnistupel
  - Alle in der **select**-Klausel aufgeführten Attributnamen müssen in der **group by**-Klausel aufgeführt werden
    - Nur so gewährleistet, dass sich Attributwerte innerhalb der Gruppe gleich

- Man erhält Tabelle von Gruppen, für die die Projektionsliste ausgewertet wird
  - Pro Gruppe ein Ergebnistupel

Elementtest

Beispiel für einen Elementtest

```
select Name
from Professoren
where PersNr in (select gelesenVon
                 from Vorlesungen)
```

```
select Name
from Professoren
where PersNr not in (select gelesenVon
                     from Vorlesungen)
```

Elementtest mit geschachtelter Anfrage häufig ersetzbar durch nichtgeschachtelte Anfrage mit Join

Quantifizierung (eingeschränkte Form)

Universelle Quantifizierung:

- $\{x \in R \mid \forall\, y \in S : x > y\}$
- Hier: Tabelle aller Projekte *x*, die ein höheres Budget als *alle* externen Projekte *y* haben

```
select *
from Projekte x
where x.Budget > all
        (select y.Budget
         from ExterneProjekte y);
```

Existentielle Quantifizierung:

- $\{x \in R \mid \exists\, y \in S : x > y\}$
- Hier: Tabelle aller Projekte *x*, die mindestens an *einer* Projektdurchführung *y* beteiligt sind

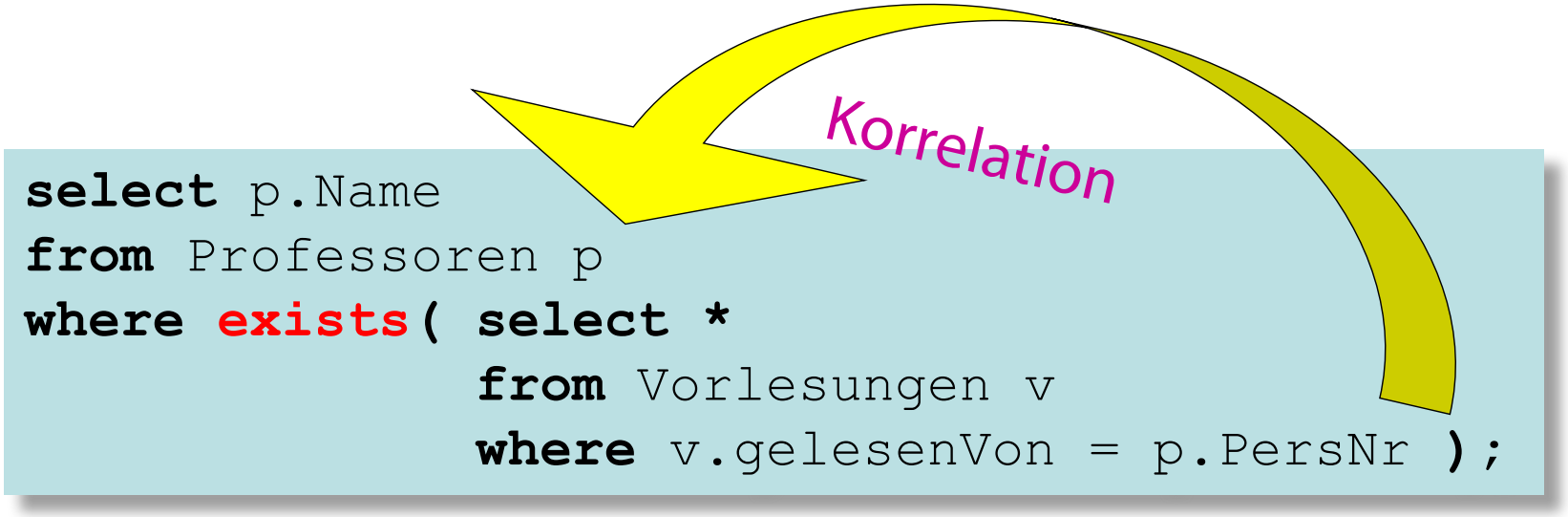
```
select *
from Projekte x
where x.Budget > any
        (select y.Budget
         from ExterneProjekte y);
```

- **= any** synonym zu **in**.

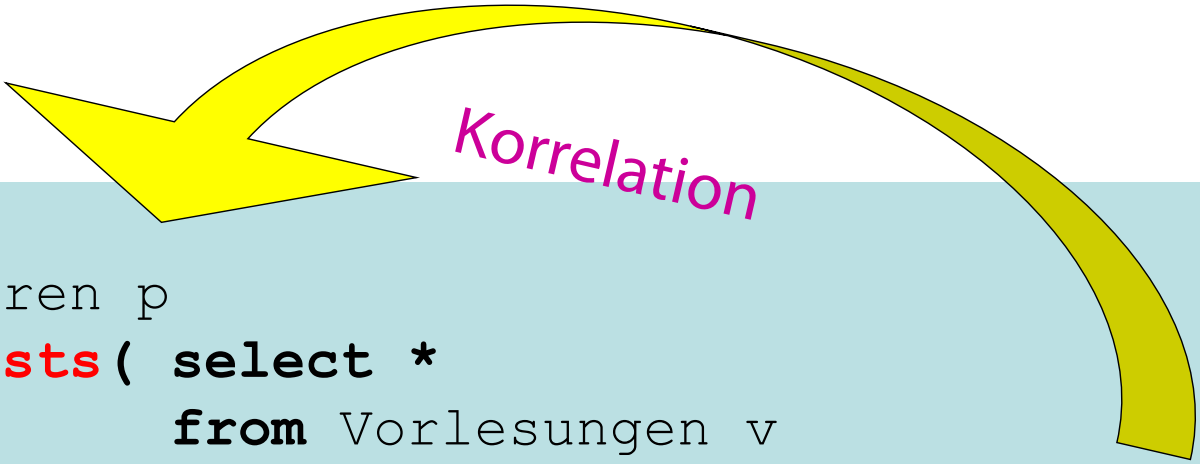
```
select *
from Projekte as x
where x.Nr = any
        (select y.Nr
         from Projektdurchfuehrungen y);
```

Quantifizierung mit **exists**

Beispiel: Liefere alle Professoren, die eine Vorlesung anbieten



Negierter Existenzquantor




```
select p.Name
from Professoren p
where not exists( select *
                  from Vorlesungen v
                  where v.gelesenVon = p.PersNr );
```

Realisierung als Mengenvergleich

Unkorrelierte Unteranfrage: meist effizienter, wird nur einmal ausgewertet

```
select Name
from Professoren
where PersNr not in ( select gelesenVon
                      from Vorlesungen );
```

Allquantifizierung

SQL-92 hat keinen Allquantor

Allquantifizierung muss also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden

Logische Formulierung der Anfrage: Wer hat **alle** vierstündigen Vorlesungen gehört?

$$\{s \mid s \in \text{Studenten} \wedge \forall v \in \text{Vorlesungen}\ (v.SWS = 4 \longrightarrow \exists h \in \text{hören}\ (h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Elimination von $\forall$ und $\longrightarrow$

Dazu sind folgende Äquivalenzen anzuwenden

$$\forall t \in R\,(P(t)) \equiv \neg(\exists t \in R(\neg P(t)))$$

$$R \longrightarrow T \equiv \neg R \vee T$$

Umformung des Kalkül-Ausdrucks ...

Wir erhalten

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen}\ \neg(\neg(v.SWS=4) \vee \exists h \in \text{hören}\ (h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Anwendung der DeMorgan-Regel ergibt schließlich:

$$\{s \mid s \in \text{Studenten} \wedge \neg(\exists v \in \text{Vorlesungen}\ (v.SWS = 4 \wedge \neg(\exists h \in \text{hören}\ (h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))))\}$$

SQL-Umsetzung folgt direkt:

```
select s.*
from Studenten s
where not exists
    (select *
      from Vorlesungen v
      where v.SWS = 4
        and not exists
            (select *
             from hören h
             where h.VorlNr = v.VorlNr
              and h.MatrNr=s.MatrNr ) );
```

Allquantifizierung durch count-Aggregation

Allquantifizierung kann auch durch eine **count-**Aggregation ausgedrückt werden

Wir betrachten dazu eine etwas einfachere Anfrage, in der wir die *(MatrNr* der) Studenten ermitteln wollen, die *alle* Vorlesungen hören:

```
select h.MatrNr
from hören h
group by h.MatrNr
having count (*) = (select count (*) from Vorlesungen);
```

Weiterverwendung von Anfragen

Definition einer Sicht (View) am Beispiel

```
create view SWUnterabteilungen as
select Name, Kurz
from Abteilungen where Oberabt = 'LTSW';
```

- Nicht das Ergebnis, sondern die Anfrage wird benannt.
- Bei jeder Verwendung wird die Basisanfrage über dem aktuellen Datenbestand ausgewertet

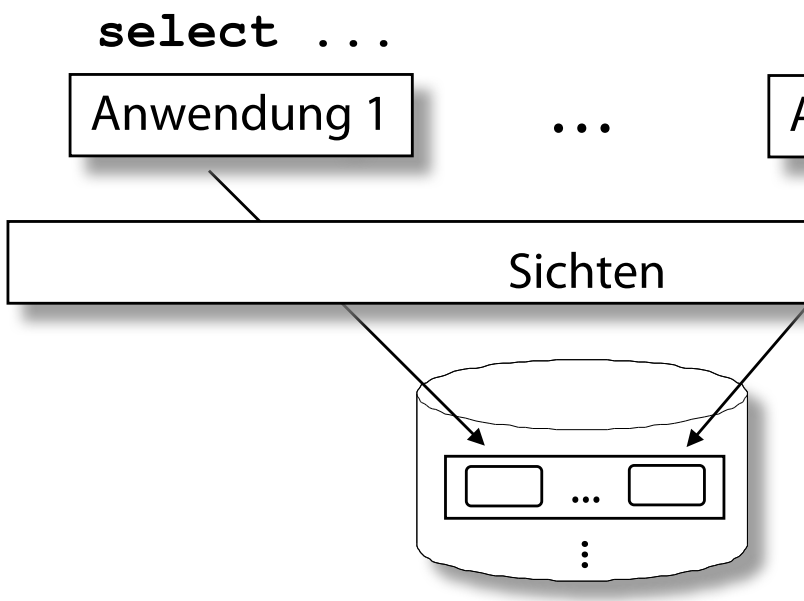
```
select u.name, p.nr
from SWUnterabteilungen u,
     Projektdurchfuehrungen p
where u.kurz = p.kurz;
```

SWUnterabteilungen wird wie eine gewöhnliche Basistabelle verwendet.

- Direkte Verwendung eines Anfrageergebnisses als Bereichsrelation einer komplexen Anfrage

```
select u.Name, p.Nr
from (select Name, Kurz
      from Abteilungen
      where Oberabt = 'LTSW') u,
     Projektdurchfuehrungen p
where u.Kurz = p.Kurz;
```

Sichten (1)



- SQL-Anfragen in der Anwendung
- SQL-View als externe Sicht
- SQL-Tabellen als konzeptuelles Schema

Projektdatenbank

(vgl. ANSI/SPARC)

Ziel:

- Kapselung der Anwendung
- Entkopplung ... (Schemaevolution)
  - Anwendung: Externe Sicht
  - DB: Konzeptuelle Sicht

```
create view ReicheProjekte
as select *
from Projekte
where Budget > 200000;
```

Änderbarkeit von Sichten

```
create view VorlesungenSicht as
select Titel, SWS, Name
from Vorlesungen, Professoren
where gelesen_von=PersNr;
```

```
insert into VorlesungenSicht
values ('Nihilismus', 2, 'Nobody');
```

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten

PersINr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Änderbarkeit von Sichten

```
create view VorlesungenSicht as
select Titel, SWS, Name
from Vorlesungen, Professoren
where gelesen_von=PersNr;
```

```
insert into VorlesungenSicht
values ('Nihilismus', 2, 'Nobody');
```

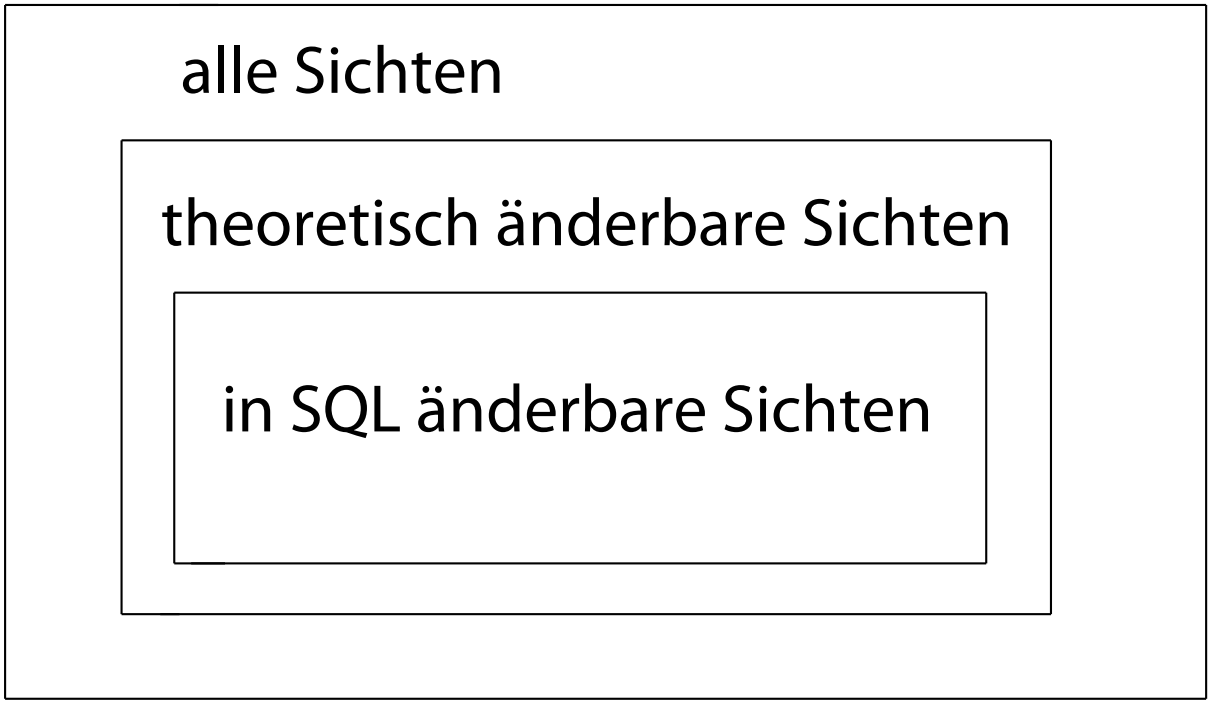
Zentrales Problem beim Datenaustausch und bei der Datenintegration

```
create view WieHartAlsPrüfer (PersNr, Durchschnittsnote) as
select PersNr, avg(Note)
from prüfen
group by PersNr;
```

Änderbarkeit von Sichten

in SQL

- nur eine Basisrelation
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



Integritätssicherung in SQL (1)

SQL-inhärente Integritätsbedingungen (⟶ *statische Typisierung*):

- Typisierung der Spalten: nur typkompatible Werte
- Tupel haben identische Spaltenstruktur

Applikationsspezifische Integritätsbedingungen

- Selbstdefinierte SQL-Domänen im aktuellen Schema

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

- Zusicherungen für Tabellen und Schemata

Integritätssicherung in SQL (2)

- Tabellenzusicherungen (Constraints)

```
create table Tabellenname (...
  constraint Zusicherungsname
    check (Prädikat)) ;

alter table add
 constraint Zusicherungsname
    check (Prädikat);
```

- Schemazusicherungen (Assertions)

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

Ein **Datenbankzustand** heißt **konsistent**, wenn alle im Schema deklarierten Zusicherungen erfüllt sind (Tabellen- und Schemazusicherungen konjunktiv verknüpft)

Spaltenwertintegrität

Tabellenzusicherung, bezogen auf Spaltennamen: Spaltenintegrität

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

Tabellenzusicherung bezogen auf Spaltennamen:
Zeilenenintegritätsbeziehung (von jeder Zeile einer Tabelle zu erfüllen)

```
check(Ausgaben <= Einnahmen)

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

Tabellenintegrität (1)

Überprüfung durch komplette mengenorientierte Anfrage:

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

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

Beschleunigung durch *Indexstrukturen* (z.B. B-Bäume, Hash-Tabelle)

⟶ *Effizienzgewinn* bei Anfragen und Änderungsoperationen führen

Tabellenintegrität (2)

Spezielle Konstrukte Für häufig auftretende Muster von Zusicherungen:

Eindeutigkeit von Spaltenwertkombinationen in einer Tabelle
(⟶ *Schlüsselkandidat*).

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

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

(x.Name = y.Name) ⟶ (x = y)

Mehrere Schlüsselkandidaten ⟶ separate unique-Klauseln

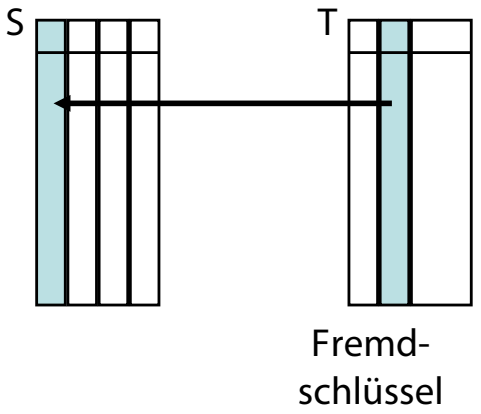
Primärschlüssel: keine Nullwerte

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

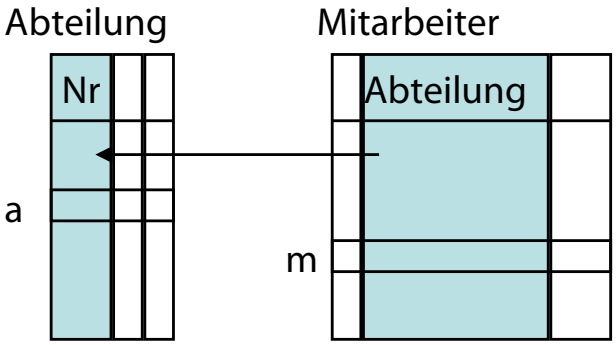
```
create table Projekte(...
  unique Nr
  check(Nr is not null))
```

Referentielle Integrität (1)

Zu jeder Reihe in Tabelle *T* existierte zugehörige Reihe in Tabelle *S*, die Fremdschlüsselwert von *T* als Wert ihres Schlüsselkandidaten besitzt.




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




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

$\forall$ m $\in$ Mitarbeiter : $\exists$ a $\in$ Abteilung: m.Abteilung = a.Nr

Referentielle Integrität (2)

Im allgemeinen besteht Fremdschlüssel einer Tabelle T aus 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, ...,B_n die Primärschlüsselspalten von S, kann ihre Angabe entfallen.

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

Behandlung von Integritätsverletzungen (1)

- Annahme: Fremdschlüsselreferenz von T nach S
- Fremdschlüsselintegrität durch vier Operationen verletzbar:
  - **insert into** T
  - **update** T **set** ...
  - **delete from** S
  - **update** S **set** ...

Behandlung von Integritätsverletzungen (2)

- Fall 1 und 2:
  - Fremdschlüsselreferenz in S evtl. nicht definiert (→ Fehler)
- Fall 3 oder 4:
  - Tupel in S gelöscht, auf das Fremdschlüsselreferenz zeigt (→ Fehler)
  - Fehlerbehandlung kann angegeben werden
    - **set null**: Der Fremdschlüsselwert aller betroffener Reihen in T durch **null** ersetzt
    - **set default**: Der Fremdschlüsselwert aller betroffener Reihen in T durch Standardwert der Fremdschlüsselspalte ersetzt
    - **cascade**:
      - Im Fall 3 (**delete**) betroffene Reihen in T gelöscht
      - Im Falle 4 (**update**) Fremdschlüsselwerte aller betroffener Reihen in T durch die **neuen** Schlüsselwerte der korrespondierenden Reihen ersetzt
    - **no action**: Anweisung zur Änderung von S wird ignoriert

Zeitpunkt der Integritätsprüfung

- Transaktionsende (deferrable)
- Nach jeder SQL-Anweisung (not deferrable)

SQL-Standardisierung

SQL-86:

- ANSI X3.135-1986 Database Language SQL, 1986
- ISO/IEC 9075:1986 Database Language SQL, 1986

SQL-89:

- ANSI X3.135-1989 Database Language SQL, 1989
- ISO/IEC 9075:1989 Database Language SQL, 1989

SQL-92:

- ANSI X3.135-1992 Database Language SQL, 1992
- ISO/IEC 9075:1992 Database Language SQL, 1992
- DIN 66315 Informationstechnik - Datenbanksprache SQL, Aug. 1993

SQL-99:

- ANSI/ISO/IEC Mehrteiliger Entwurf: Database Language SQL
- ANSI/ISO/IEC 9075:1999: Verabschiedung der Teile 1 bis 5
  9075:2000: Teil 10 9075:2001: Teil 9

SQL-2003/06:

- ANSI/ISO/IEC Integration von XML in SQL

SQL-2011:

- SQL:2011 or ISO/IEC 9075:2011 aktuelle Version

Überblick

Der Standard besteht insgesamt aus 9 einzelnen Publikationen:[3]

- ISO/IEC 9075-1:2016 Part 1: Framework (SQL/Framework)
- ISO/IEC 9075-2:2016 Part 2: Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2016 Part 3: Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2016 Part 4: Persistent stored modules (SQL/PSM)
- ISO/IEC 9075-9:2016 Part 9: Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2016 Part 10: Object language bindings (SQL/OLB)
- ISO/IEC 9075-11:2016 Part 11: Information and definition schemas (SQL/Schemata)
- ISO/IEC 9075-13:2016 Part 13: SQL Routines and types using the Java TM programming language (SQL/JRT)
- ISO/IEC 9075-14:2016 Part 14: XML-Related Specifications (SQL/XML)

und wird durch 6 bzw. 7 ebenfalls standardisierte *SQL multimedia and application packages* ergänzt:

- ISO/IEC 13249-1:2016 Part 1: Framework
- ISO/IEC 13249-2:2003 Part 2: Full-Text
- ISO/IEC 13249-3:2016 Part 3: Spatial
- ISO/IEC 13249-5:2003 Part 5: Still image
- ISO/IEC 13249-6:2006 Part 6: Data mining
- ISO/IEC 13249-7:2013 Part 7: History
- ISO/IEC 13249-8:xxxx Part 8: Metadata registries (MDR) (noch nicht verabschiedet)

Der offizielle Standard ist nicht frei verfügbar, jedoch existiert ein Zip-Archiv mit einer Arbeitsversion von 2008.[4]