Datenbanken

SQL (Teil 1)

Dr. Özgür Özçep

Universität zu Lübeck

Institut für Informationssysteme

RDM: Projektdatenbank

Projekte

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

Projektdurchführung

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

Abteilungen

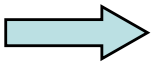
Kurz	*Name*	*Oberabt*
MFSW	Mainframe SW	LTSW
UXSW	Unix SW	LTSW
PCSW	PC SW	LTSW
LTSW	Leitung SW	**NULL**
PERS	Personal	**NULL**

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';
```

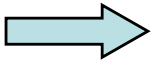


Ergebnistabelle

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)

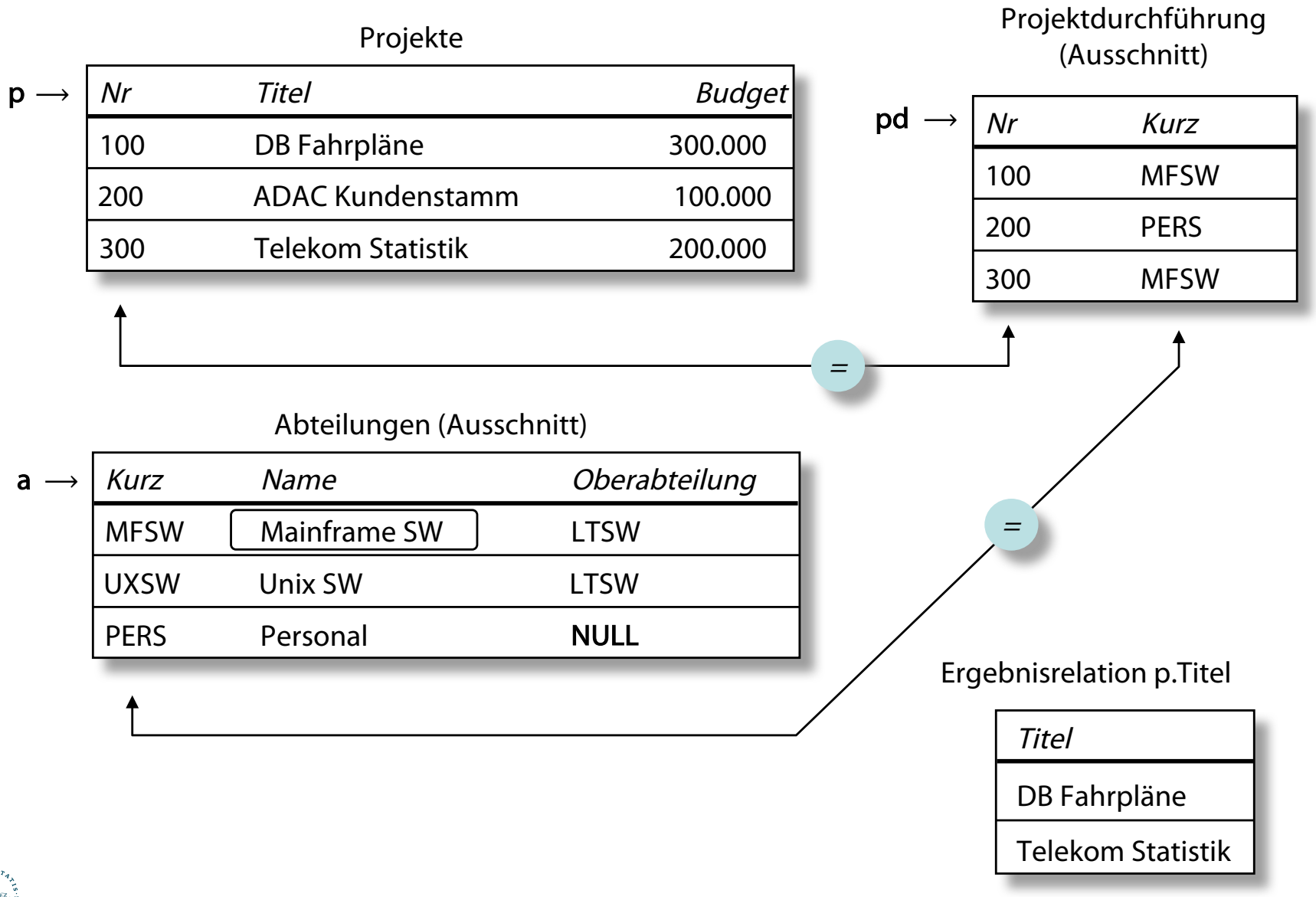
Iterationsabstraktion mit Hilfe des **select from where**-Konstrukts:

- **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



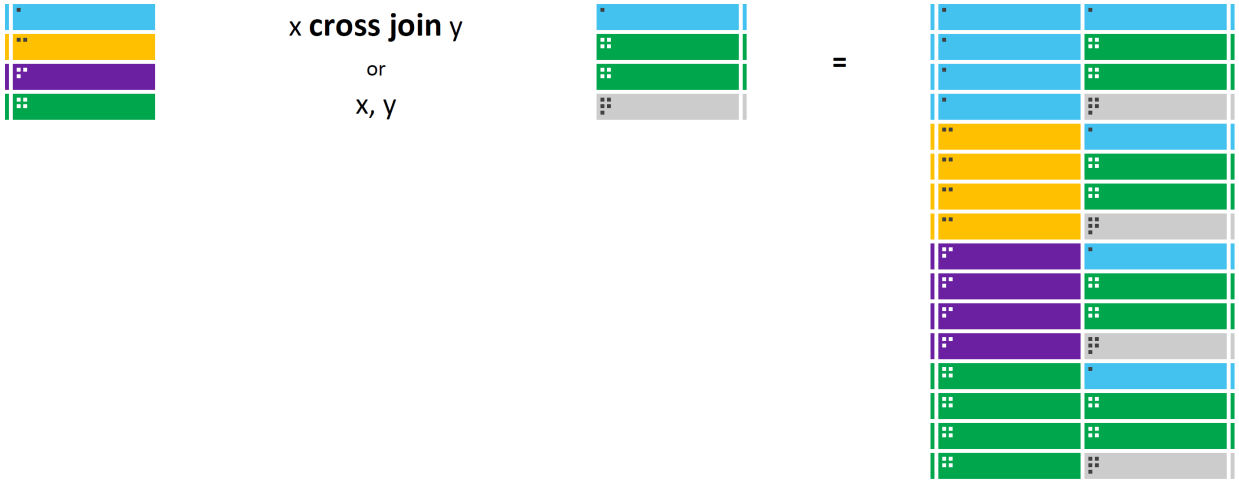
Vermeidung von Variablen

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

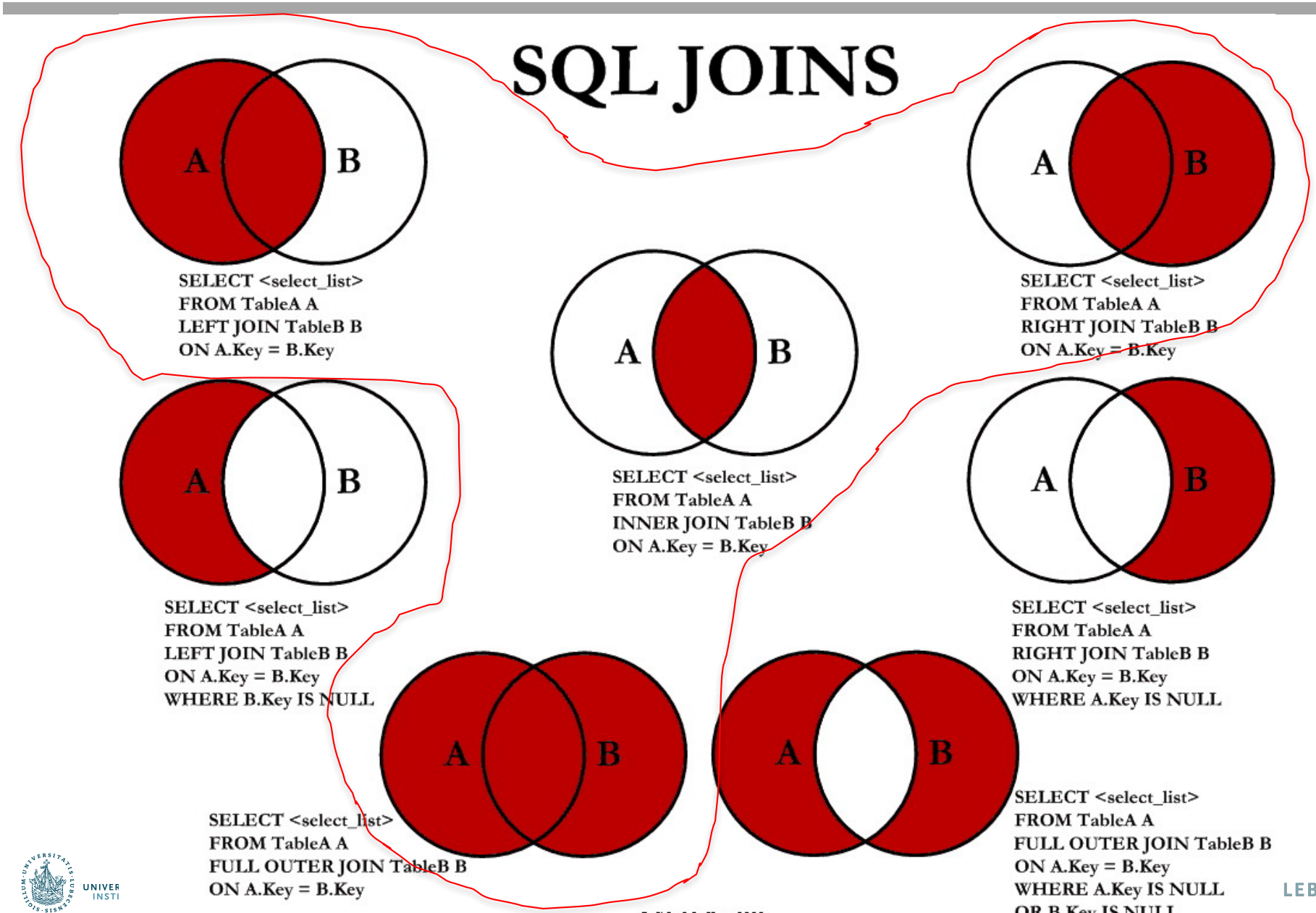
Join-Operatoren:

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

SQL Join Types



Zum Verständnis der Namensgebung...



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 (user defined functions, UDFs) 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 _
- **Schlüsselwörter**: SQL definiert über 210 Namen als Schlüsselwörter, die nicht kontextsensitiv sind
- **Begrenzte Namen**: Zeichenketten in doppelten Anführungszeichen (nützlich zur Verwendung von Schlüsselwörtern 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;
```

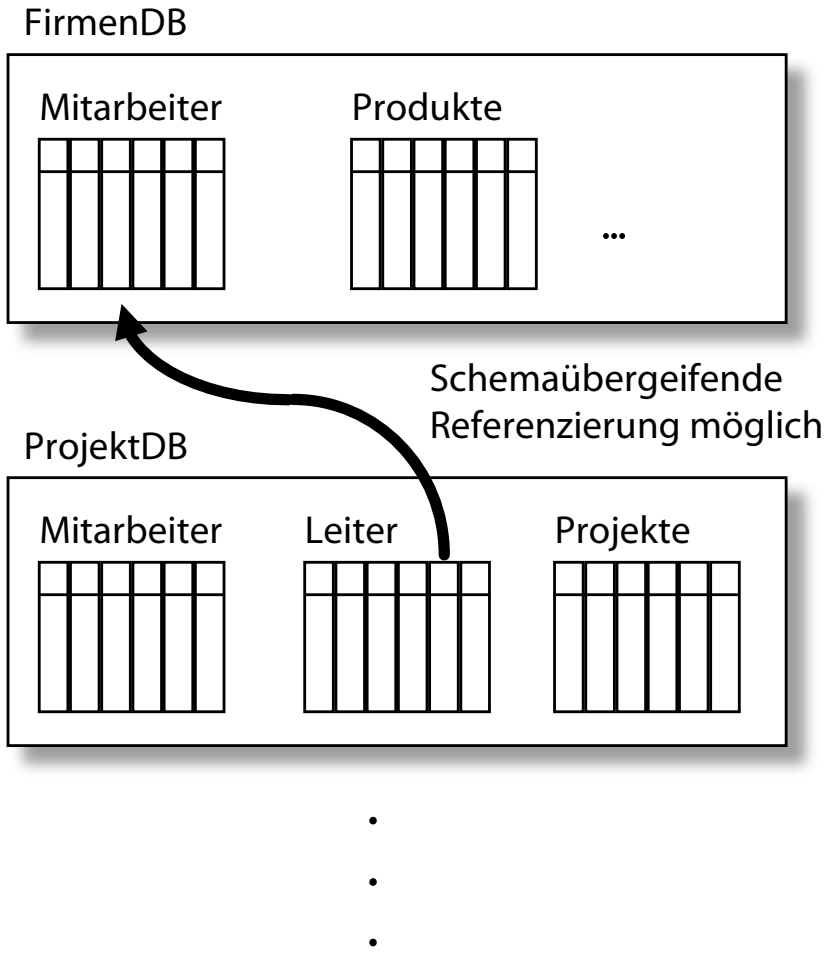
- Schemata werden persistent gespeichert (zugreifbar über SQL)
- Multiple Schemata nötig für:
  - Integration separat entwickelter Datenbanken
  - Arbeit in verteilten und föderativen Datenbanken

Schemata und Kataloge (2)

Schemakatalog

Name	Benutzer	
FirmenDB	matthes	
ProjektDB	matthes	
TextDB	schmidt	

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



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
```

- 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*).

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

Annahmen

- Closed-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

- Closed-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 wird 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, ...)
- Semantik nicht vereinbar mit „Certain Answer Semantics“

„Multiple issues with SQL's handling of nulls have been well documented. Having efficiency as its key goal, evaluation of SQL queries disregards the standard notion of correctness on incomplete databases – certain answers – due to its high complexity. As a result, it may produce answers that are just plain wrong"

Guagliardo/Libkin: **Correctness of SQL Queries on Databases with Nulls, PODS 2016.**

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**

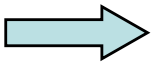
Abteilungen

Projektdatenbank

Duplikatelimination

Elimination von Duplikaten im Anfrageergebnis mit dem Schlüsselwort distinct:

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



Oberabt
LTSW
NULL

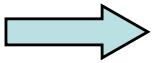
Hier: Umwandlung einer Ergebnistabelle in *Ergebnismenge*

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;
```

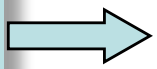


Oberabt
LTSW

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;
```

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**

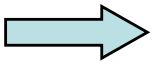
Abteilungen

Projektdatenbank

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;
```

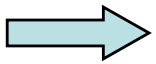


p.Budget

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)

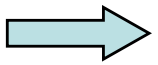
```
select count(*)
from Abteilungen;
```



count(*)
5

Einmaliges Zählen von Werten möglich (nur Nicht-Nullwerte)

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

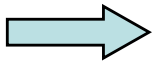


count(*)
1

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

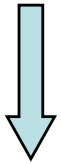
sum	*max*
700.000	300.000

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

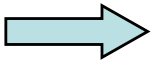
Gruppierung: Beispiel

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" - da mit select ausgegeben - muss in "group by" 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 Attributwerte innerhalb der Gruppe gleich

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