

**(Bisherige) Gliederung der Vorlesung**

- »»»0. Vorbemerkungen
- »»»1. Einführung in Datenbanken
- »»»2. Datenbankmodelle für den Entwurf
- »»»3. Datenbankmodelle für die Realisierung
- »»»4. Grundlagen von Anfragen und Änderungen
  - »»»4.1 Kriterien für Anfragesprachen
  - »»»4.2 Relationenalgebra
  - »»»4.3 Anfragekalküle
    - »»»4.3.1 Bereichskalkül
    - »»»4.3.2 Tupelkalkül
- »»»5. Datenbanksprache SQL

**Datenbanksprache SQL**

**SQL (Structured Query Language)** wurde am IBM San Jose Research Laboratory entwickelt. Die Entwicklung von Standard-SQL erfolgte in den folgenden Etappen:

- bis Anfang der 80er Jahre Wildwuchs verschiedener Dialekte
- erste Sprachstandardfestlegung um 1986
- endgültiger Sprachstandard im Jahre 1989 = SQL-89
- in Deutschland als DIN ISO 9075 im Jahre 1990 publiziert
- inzwischen gibt es aus dem Jahre 1992 **SQL-92 = SQL 2**
- an SQL 3 / SQL 4 wird gearbeitet
- SQL-99 für „objekt-relationale Datenbanken“ standardisiert



## Datenbanksprache SQL

umfaßt Sprachanteile

- zur Definition von Datenbankschemata  
(DDL - Data Definition Language)
- zur Formulierung von Anfragen  
(DQL - Data Query Language)
- zur Formulierung von Datenbankänderungsoperationen  
(DML - Data Manipulation Language)



## 5.1. SQL - DDL

### Anforderungen an eine relationale DDL

nach Codd 1982 Sprachmittel zur Definition von

1. Attributen
2. Wertebereichen
3. Relationenschemata
4. Primärschlüsseln
5. Fremdschlüsseln



## SQL - DDL Sprachkonstrukte

- create table**
- alter table**
- drop table**

*in SQL-92 auch*

- create domain**
- alter domain**
- drop domain**



## Die Anweisung create table

```
create table basisrelationenname  
    (spaltenname_1 wertebereich_1 [not null],  
    ...  
    spaltenname_k wertebereich_k [not null])
```

**Erlaubte Wertebereiche in create table**

- integer* (oder auch *integer4*, *int*)
- smallint* (oder auch *integer2*)
- float(p)* (oder auch kurz *float*)
- decimal(p,q)* und *numeric(p,q)* mit jeweils *q* Nachkommastellen
- character(n)* (oder kurz *char(n)*, bei  $n = 1$  auch *char*) für Strings fester Länge *n*
- character varying(n)* (oder kurz *varchar(n)*) für Strings variabler Länge bis zur Maximallänge *n*
- bit(n)* oder *bit varying(n)* analog für Bitfolgen
- date*, *time* bzw. *timestamp* für Datums-, Zeit- und kombinierte Datums-Zeit-Angaben

**Die Anweisung create table**

Mit **not null** können in bestimmten Spalten *Nullwerte* als Attributwerte ausgeschlossen werden:

```
create table Bücher  
    ( ISBN char(10) not null,  
      Titel varchar(200),  
      Verlagsname varchar(30) )
```

**SQL-89 Level 2 mit IEF**

- zweite Stufe der SQL-89-Norm sieht Zusatz IEF (Integrity Enhancement Feature) vor
- Definition von Schlüsseln und Fremdschlüsseln

**Beispiel Tabellendefinition mit IEF**

```
create table Bücher
  ( ISBN char(10) not null,
    Titel varchar(200),
    Verlagsname varchar(30),
    primary key (ISBN),
    foreign key (Verlagsname)
      references Verlage (Verlagsname) )
```

**create table in SQL-92**

```
create table Bücher
  ( ISBN char(10),
    Titel varchar(200),
    Verlagsname varchar(30),
    primary key (ISBN),
    foreign key (Verlagsname)
      references Verlage (Verlagsname) )
```

**not null** implizit durch die **primary key**-Klausel

**Erweiterungen in SQL-92:**

- Definition von Schlüsseln und Fremdschlüsseln
- default**-Klausel: Defaultwerte für Attribute
- create domain**-Anweisung für benutzerdefinierte Wertebereiche
- check**-Klausel zur Festlegung von weiteren lokalen Integritätsbedingungen innerhalb der zu definierenden Wertebereiche, Attribute und Relationenschemata



### Definition eines Wertebereichs

```
create domain Gebiete varchar(20)  
default 'Informatik'  
  
create table Vorlesungen  
(V_Bezeichnung varchar(80) not null,  
SWS smallint,  
Semester smallint,  
Studiengang Gebiete )  
  
create table Mitarbeiter  
(PANr integer not null,  
AngNr char(10) not null,  
Fachbereich Gebiete,  
Gehalt decimal(10,2),  
Raum integer,  
Einstellung date )
```



### Integritätsbedingungen mit check-Klausel in Wertebereichsdefinitionen

```
create domain Gebiete varchar(20) default 'Informatik'  
check ( value in ('Informatik', 'Mathematik', 'Elektrotechnik', 'Linguistik') )
```



## Integritätsbedingungen mit check-Klausel in Attributdefinitionen

```
create table Vorlesungen
```

```
(V_Bezeichnung varchar(80) primary key,
```

```
SWS smallint check (SWS ≥ 0),
```

```
Semester smallint check (Semester between 1 and 9),
```

```
Studiengang Gebiete )
```



## Integritätsbedingungen mit check-Klausel für eine Tabelle

```
create table Buch_Versionen
```

```
( ISBN char(10),
```

```
Auflage smallint check (Auflage > 0),
```

```
Jahr integer check (Jahr between 1800 and 2020),
```

```
Seiten integer check (Seiten > 0),
```

```
Preis decimal(8,2) check (Preis ≤ 250),
```

```
primary key (ISBN, Auflage),
```

```
foreign key (ISBN) references Bücher (ISBN),
```

```
check ((select sum(Preis) from Buch_Versionen) <
```

```
(select sum(Budget) from Lehrstühle)))
```

Hinweis:

Erläuterung der select-  
Anweisung kommt später!

**Die Anweisungen alter table und drop table**

Syntax des **alter table**-Kommandos in SQL-89:

```
alter table basisrelationenname  
           add spaltenname wertebereich
```

```
alter table Lehrstühle  
           add Budget decimal(8,2)
```

Wirkung ist:

- Änderung des Relationenschemas im Data Dictionary (ein neues Attribut wird dem Relationenschema Lehrstühle zugeordnet)
- Erweiterung der existierenden Basisrelation um ein Attribut, das bei jedem existierenden Tupel mit **null** besetzt wird.

**alter table-Kommando in SQL-2**

Statt

```
add spaltenname wertebereich
```

auch Angabe von Default-Werten und **check**-Klauseln erlaubt:

```
add Budget decimal(8,2) default 10000  
           check (Budget > Anzahl_Planstellen x 1000)
```

**alter-und drop-Klausel für Attribute (Spaltennamen)** Die Klausel

**alter** spaltenname default\_änderung  
nur Änderung der Defaultwerte, nicht Änderung von Datentypen

 Die Klausel

**drop** spaltenname { **restrict** | **cascade** }  
erlaubt Löschen von Attributen, falls  
- keine Integritätsbedingungen mit Hilfe dieses Attributs definiert wurden (im Fall **restrict**)  
- oder mit gleichzeitiger Löschung dieser Integritätsbedingungen (im Fall **cascade**)

**Die Anweisung drop table**

**drop table** basisrelationenname { **restrict** | **cascade** }

**Wirkung:**

- Basisrelationenschema wird aus dem Data Dictionary entfernt und damit auch die Relation aus der Datenbank
- restrict** und **cascade** analog zu **drop** bei Attributen

**5.2. SQL-Anfragesprache**

**SQL-Anfragen** bestehen im wesentlichen aus dem sogenannten SFW-Block (für select ... from ... where ...)

**select**

- bestimmt Projektionen auf Spalten mehrerer Tabellen

**from**

- legt zu verwendende (Basis-)Relationen fest

**where**

- definiert Selektions- und Verbundbedingungen

**group by**

- Auswertung auf Untergruppen (Aggregatfunktionen)

**having**

- Selektionsbedingungen für Bildung von Gruppen

**Beispiele mit Büchern wieder aufgegriffen:****Autoren**

ISBN	Autor
0-8053-1753-8	Elmasri
0-8053-1753-8	Navathe
3-8244-2021-X	Schürr
3-8244-2075-9	Zündorf

**Ausleihe**

ISBN	Name
0-8053-1753-8	Schmitz
0-8053-1753-8	Derichsweiler
3-8244-2021-X	Radermacher
3-8244-2075-9	Radermacher

**Bücher**

ISBN	Titel	Verlag
0-8053-1753-8	Principles of Database Systems	Benj. Cummings
3-8244-2021-X	Operationales Spezifizieren mit ...	Deutscher Universitäts-Verlag
3-8244-2075-9	PROgrammierte GraphERsetzungsSysteme	Deutscher Universitäts-Verlag



**from-Klausel**

Syntax

```
select *  
from relationenliste
```

Beispiel

```
select *  
from Bücher
```

liefert die gesamte Relation Bücher



**Kartesisches Produkt von zwei Relationen:**

```
select * from Autoren, Ausleihe
```

⇐ bildet kartesisches Produkt der beiden Tabellen und selektiert mit \* alle Spalten der resultierenden Tabelle

Autoren.ISBN	Autor	Ausleihe.ISBN	Name
0-8053-1753-8	Elmasri	0-8053-1753-8	Schmitz
0-8053-1753-8	Navathe	0-8053-1753-8	Schmitz
3-8244-2021-X	Schürr	0-8053-1753-8	Schmitz
3-8244-2075-9	Zündorf	0-8053-1753-8	Schmitz
0-8053-1753-8	Elmasri	0-8053-1753-8	Derichsweiler
0-8053-1753-8	Navathe	0-8053-1753-8	Derichsweiler
3-8244-2021-X	Schürr	0-8053-1753-8	Derichsweiler
3-8244-2075-9	Zündorf	0-8053-1753-8	Derichsweiler
0-8053-1753-8	Elmasri	3-8244-2021-X	Radermacher
...	...	...	...

**Kartesisches Produkt**

Einführung von Tupelvariablen um z.B. auf eine Relation mehrfach zugreifen zu können

```
select * from Bücher eins , Bücher zwei
```

Ergebnis hat sechs Spalten:

eins.ISBN, eins.Titel, eins.Verlag, zwei.ISBN, zwei.Titel, zwei.Verlag

*Selbst-Verbund* (Self-Join) für tupelübergreifende Selektionen

**Bezug zum Tupelkalkül**

Korrespondierende Kalkülausdrücke automatisch sicher

```
select ...  
from Bücher eins , Bücher zwei  
where ...
```

$\{ \dots | \text{Bücher}(\text{eins}) \wedge \text{Bücher}(\text{zwei}) \wedge (\dots) \}$

**SQL - Anfrage mit Verbundbildung:**

```
select Autoren.ISBN, Autor, Name
```

```
from Autoren, Ausleihe
```

```
where Autoren.ISBN = Ausleihe.ISBN
```

⇨ berechnet den Verbund der beiden Tabellen über ISBN (natural join, natürlicher Verbund)

⇨ anstelle von "=" geht auch Vergleich auf "<>" oder "<" oder ...

( $\theta$ -join,  $\theta$ -Verbund)

Autoren.ISBN	Autor	Name
0-8053-1753-8	Elmasri	Schmitz
0-8053-1753-8	Navathe	Schmitz
0-8053-1753-8	Elmasri	Derichsweiler
0-8053-1753-8	Navathe	Derichsweiler
3-8244-2021-X	Schürr	Radermacher
3-8244-2075-9	Zündorf	Radermacher

**SQL - 92 - Spezialitäten**

Verbunde als explizite Operatoren

kartesisches Produkt

```
select *
from Bücher, Ausleihe
```

```
select *
from Bücher cross join Ausleihe
```

Verbund über Verbundbedingungen

```
select *
from Bücher, Ausleihe
where Bücher.ISBN = Ausleihe.ISBN
```

join-Operator:  $\theta$ -Verbund

```
select *
from Bücher join Ausleihe
on Bücher.ISBN = Ausleihe.ISBN
```

**Weitere Verbunde in SQL-92**

## Gleichverbund

```

select *
from Bücher join Ausleihe
using (ISBN)

```

## natürlicher Verbund

```

select *
from Bücher natural join Ausleihe

```



Statt **inner join** nun **outer join** (dangling tuples übernehmen und mit Nullwerten auffüllen)

**full outer join:** in beiden Operanden

**left outer join:** im linken Operanden

**right outer join:** im rechten Operanden

LINKS

A	B
1	2
2	3

RECHTS

B	C
3	4
4	5

NATURAL JOIN

A	B	C
2	3	4

OUTER

A	B	C
1	2	⊥
2	3	4
⊥	4	5

LEFT

A	B	C
1	2	⊥
2	3	4

RIGHT

A	B	C
⊥	4	5
2	3	4



### Die select-Klausel

Relationenalgebra: abschliessende Projektion

Relationenkalkül: Zielliste

```
select [distinct] {attribut |  
                  arithmetischer-ausdruck |  
                  aggregat-funktion}  
  
from ...
```

- Attribute aus **from**-Relationen
- Arithmetische Ausdrücke über Attributen und Konstanten
- Aggregatfunktionen über Attributen

**distinct**: Ergebnismenge statt Multimenge



### Projektionsergebnis Menge oder Multimenge

**select** Name **from** Ausleihe

Name
Schmitz
Derichsweiler
Radermacher
Radermacher

**select distinct** Name **from** Ausleihe

Name
Schmitz
Derichsweiler
Radermacher

**Tupelvariablen und Relationennamen**

Angabe der Attributnamen durch Präfix ergänzen, falls nicht eindeutig:

und

```
select ISBN from Bücher
```

```
select Bücher.ISBN from Bücher
```

sind äquivalent

```
select ISBN, Titel, Name      (falsch!)
from Bücher, Ausleihe
where Bücher.ISBN = Ausleihe.ISBN
```

```
select Bücher.ISBN, Titel, Name (richtig)
from Bücher, Ausleihe
where Bücher.ISBN = Ausleihe.ISBN
```

Tupelvariable kann benutzt werden:

```
select eins.ISBN, zwei.Titel
from Bücher eins, Bücher zwei
```

**Bezug zum Tupelkalkül**

**select:** Zielliste im Tupelkalkül

Letzte Anfrage entspricht

$$\{ \text{eins.ISBN, zwei.Titel} \mid \text{Bücher (eins)} \wedge \text{Bücher (zwei)} \}$$



## Die where-Klausel

Selektionsbedingung der Relationenalgebra oder Verbundbedingung

**select ... from ... where** bedingung

Bedingung:

- Konstanten-Selektion  
attribut  $\theta$  konstante
- Attribut-Selektion zwischen zwei Attributen mit kompatiblen Wertebereichen:  
attribut1  $\theta$  attribut2
- weitere Bedingungen auf den folgenden Seiten



## Verbundbedingung

relation1.attribut = relation2.attribut

Beispiel: natürlicher Verbund

```
select Bücher.ISBN, Titel, Name  
from Bücher, Ausleihe  
where Bücher.ISBN= Ausleihe.ISBN
```

auch Gleichverbund und  $\theta$  -Verbund erlaubt

**Bereichsselektion**

attribut **between** konstante1 **and** konstante2

Abkürzung für

attribut  $\geq$  konstante1 **and** attribut  $\leq$  konstante2

Beispiel

```
select ISBN  
from Autoren  
where Autor between ‚A‘ and ‚N‘
```

**Ungewißheitsselektion**

theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung

attribut **like** spezialkonstante

Spezialkonstante kann '%' (kein oder beliebig viele Zeichen) oder '\_' (genau ein Zeichen) beinhalten

Anwendung: Selektion nach Büchern von Benjamin/Cummings

```
select * from Bücher  
where Verlagsname like `Benj%Cummings`
```

ist Abkürzung für

```
select * from Bücher  
where Verlagsname = `Benjamin Cummings`  
or Verlagsname = `Benjamin/Cummings`  
or Verlagsname = `Benjamin-Cummings`  
or Verlagsname = `Benjamin and Cummings`  
or Verlagsname = `BenjXFDGYWCummingsSCHlumpf`  
or ...
```



## Weitere Bedingungen

*Null*-Selektion

attribut **is null**

Quantifizierte Bedingungen , wenn ein Argument in Vergleich Menge liefert

**(all, any, some und exists)**

boolesche Ausdrücke mit Konnektoren

**or, and und not**



## Bezug zum Tupelkalkül

**where**-Klausel: qualifizierende Formel in Tupelkalkülanfragen

```
select Titel, Name  
from Bücher, Ausleihe  
where Bücher.ISBN = Ausleihe.ISBN and  
      (Verlagsname = `Thomson` or Verlagsname = `ITP`);
```

entspricht

$$\{ b.\text{Titel}, a.\text{Name} \mid \text{Bücher}(b) \wedge \text{Ausleihe}(a) \wedge \\ b.\text{ISBN} = a.\text{ISBN} \wedge \\ (b.\text{Verlagsname} = \text{'Thomson'} \vee b.\text{Verlagsname} = \text{'ITP'}) \}$$



## Schachtelung von Anfragen

**where**-Klausel kann geschachtelt werden

SFW-Blöcke liefern im allgemeinen mehrere Werte

Vergleiche mit Wertemengen

- Standardvergleiche in Verbindung mit Quantoren **all** ( $\forall$ ) oder **any** ( $\exists$ )
- spezielle Prädikate für den Zugriff auf Mengen **in** und **exists**



## Das in-Prädikat und geschachtelte Anfragen

attribut **in** ( SFW-block )

Beispiel:

```
select Titel from Bücher
where ISBN in ( select ISBN from Ausleihe )
```

natürlicher Verbund mit nachfolgender Projektion

Abarbeitung:

- Ergebnis der inneren **select**-Anweisung hinter **in** als Liste von Konstanten einsetzen
- Dann modifizierte Anfrage

```
select Titel from Bücher
where ISBN in
( '0-8053-1753-8', '3-8244-2021-X', '3-8244-2075-9' )
abarbeiten
```



## umfangreicheres Beispiel einer relationalen Datenbank: Datenbank2

Personen	PANr	Vorname	Nachname	PLZ	Ort	Straße	HNr	Geb.datum
	4711	Andreas	Heuer	18209	DBR	BHS	15	31.10.1958
	5588	Gunter	Saake	39106	MD	STS	55	05.10.1960
	6834	Michael	Korn	39104	MD	BS	41	24.09.1974
	7754	Andreas	Möller	18209	DBR	RS	31	25.02.1976
	8832	Tamara	Jagellovsk	38106	BS	GS	12	11.11.1973
	9912	Antje	Hellhof	18059	HRO	AES	21	04.04.1970
	9999	Christa	Loeser	69121	HD	TS	38	10.05.1969

Pers_Telefon	PANr	Telefon
	4711	038203-12230
	4711	0381-498-3401
	4711	0381-498-3427
	5588	0391-345677
	5588	0391-5592-3800
	9999	06221-400177



Mitarbeiter	PANr	AngNr	Fachbereich	Gehalt	Raum	Einstellung
	4711	HRO-	Informatik	6000	209	01.03.94
	5588	MD-5267	Informatik	6000	304	01.04.94
	6834	MD-	Mathemati	750	309	01.09.94
	7754	HRO-	Informatik	550	218	01.10.94
	8832	MD-4567	Informatik	2800	302	01.08.94
	9912	HRO-	Linguistik	2600	008	01.01.93

Professoren	PANr	Lehrstuhlbezeichnung	Stufe
	4711	Datenbank- und Informationssysteme	C4
	5588	Datenbanken und Informationssysteme	C4

Studenten	PANr	Matrikelnummer	Studienfach	Immatrikulationsdatum
	6834	MD-891372	Informatik	01.10.89
	7754	HRO-912291	Informatik	01.10.91



## 5. Datenbanksprache SQL

## Lehrstühle

Lehrstuhlbezeichnung	Anzahl_Planstellen
Datenbank- und Informationssysteme	4
Datenbank und Informationssysteme	5
Rechnernetze	2

## Vorlesungen

V_Bezeichnung	SWS	Semester	Studiengang
Datenbanken I	4	5	Informatik
Datenbanken II	4	6	Informatik
Datenbanken	3	7	Mathematik
Objektorientierte Datenbanken	4	6	Informatik
Datenbanken für Ingenieure	2	7	Elektrotechnik
Verteilte Datenbanken	2	8	Informatik
Theorie relationaler Datenbanken	3	9	Informatik
Spezifikationsmethoden	3	10	Informatik



## 5. Datenbanksprache SQL

## Bücher

ISBN	Titel	Verlagsname
3-89319-175-5	Das DB2-Handbuch	Addison-Wesley
0-8053-1753-8	Princ. of DBS	Benj./Cummings
0-201-53771-0	Foundations of DB	Addison-Wesley
3-929821-31-1	Datenbanken	Thomson

## Buch\_Autor

ISBN	Autor
3-89319-175-5	Vossen
3-89319-175-5	Witt
0-8053-1753-8	Elmasri
0-8053-1753-8	Navathe
0-201-53771-0	Abiteboul
0-201-53771-0	Hull
0-201-53771-0	Vianu

## Buch\_Stichwort

ISBN	Stichwort
3-89319-175-5	RDB
0-8053-1753-8	RDB
0-8053-1753-8	Lehrbuch
0-8053-1753-8	ER
0-201-53771-0	RDB
0-201-53771-0	Theorie
3-929821-31-1	RDB



### 5. Datenbanksprache SQL

Buch_Versionen	ISBN	Auflage	Jahr	Seiten	Preis
	3-89319-175-5	1	1990	288	79,-
	0-8053-1753-8	1	1989	802	72,35
	0-8053-1753-8	2	1994	873	88,85
	0-201-53771-0	1	1995	685	87,45
	3-929821-31-1	1	1995	5**	79,-

Buch_Exemplare	Inventarnr	ISBN	Auflage
	001	3-89319-175-5	1
	005	0-8053-1753-3	1
	084	0-8053-1753-3	2
	085	0-8053-1753-3	2
	101	0-201-53771-0	1
	101	0-201-53771-0	1
	138	3-929821-31-1	1
	139	3-929821-31-1	1
	140	3-929821-31-1	1
	141	3-929821-31-1	1



### 5. Datenbanksprache SQL

Verlage	Verlagsname	Verlagsort
	Addison-Wesley	Bnn
	Benjamin/Cummings	Redwood City
	Thomson	Bonn



## 5. Datenbanksprache SQL

Ausleihe

PANr	Inventarnr
7754	001
7754	140
4711	141
5588	101
9912	102
9912	085

Prüft

PANr	Matrikelnummer	V_Bezeichnung	Note
4711	HRO-912291	Datenbanken I	2.0
4711	HRO-912291	Objektorientierte Datenbanken	2.3
5588	MD-891372	Datenbanken I	1.3
5588	MD-891372	Spezifikationsmethoden	2.7



## 5. Datenbanksprache SQL

Empfiehl

PANr	ISBN	V_Bezeichnung
4711	3-929821-31-1	Datenbanken I
4711	0-8053-1753-8	Datenbanken I
4711	0-8053-1753-8	Datenbanken II
4711	0-201-53771-0	Theorie relationaler Datenbanken
5588	3-89319-175-5	Datenbanken für Ingenieure
5588	0-8053-1753-8	Datenbanken I
5588	0-8053-1753-8	Datenbanken II
5588	3-929821-31-1	Datenbanken I

Vorl\_Voraus

V_Bezeichnung	Voraussetzung
Datenbanken II	Datenbanken I
Objektorientierte Datenbanken	Datenbanken I
Spezifikationsmethoden	Objektorientierte Datenbanken
Spezifikationsmethoden	Theorie relationaler Datenbanken
Theorie relationaler Datenbanken	Datenbanken I
Verteilte Datenbanken	Datenbanken II
Verteilte Datenbanken	Objektorientierte Datenbanken



Liest	PANr	V_Bezeichnung	Semester
	4711	Datenbanken I	WS 95/96
	4711	Theorie relationaler Datenbanken	WS 95/96
	4711	Datenbanken II	SS 96
	4711	Objektorientierte Datenbanken	SS 96
	5588	Datenbanken für Ingenieure	WS 95/96
	5588	Datenbanken I	WS 95/96
	5588	Verteilte Datenbanken	SS 96
	5588	Spezifikationsmethoden	SS 96

Hört	Matrikelnummer	V_Bezeichnung	Semester
	MD-891372	Datenbanken I	5
	MD-891372	Spezifikationsmethoden	6
	MD-891372	Verteilte Datenbanken	6
	HRO-912291	Datenbanken I	5
	HRO-912291	Objektorientierte Datenbanken	6
	HRO-912291	Datenbanken II	8



### Verzahnt geschachtelte Anfragen

in der inneren Anfrage Relationen- oder Tupelvariablen-Name aus dem **from**-Teil der äußeren Anfrage verwenden

Beispiel: „Nachnamen von Prüfern, die schon einmal eine 1,0 als Note gegeben haben“

```
select Nachname
from Personen
where 1.0 in ( select Note
              from Prüft
              where PANr = Personen.PANr)
```



### Verzahnt geschachtelte Anfragen

- in der äußeren Anfrage das erste Personen-Tupel untersuchen
- Ergebnis in innere Anfrage einsetzen
- innere Anfrage

```
select Note
from Prüft
where PANr = 4711
```

auswerten, liefert Werteliste ( 2.0, 2.3 )
- Ergebnis der inneren Anfrage in die äußere einsetzen  
1.0 **in** ( 2.0, 2.3 ) ergibt **false**, also ersten Prüfer nicht berücksichtigen
- in der äußeren Anfrage das zweite Personen-Tupel untersuchen usw.



### Das exists-Prädikat

testet, ob Ergebnis der inneren Anfrage nicht leer

Beispiel: „ISBNs der zur Zeit ausgeliehenen Bücher“

```
select ISBN
from Buch_Exemplare
where exists
( select *
  from Ausleihe
  where Inventarnr = Buch_Exemplare.Inventarnr)
```

**Beispiel-Datenbank**

Ausleihe	PANr	Inventarnr
	7754	001
	7754	140
	4711	141
	5588	101
	9912	102
	9912	085

  

Buch_Exemplar	Inventarnr	ISBN	Auflage
	001	3-89319-175-5	1
	005	0-8053-1753-3	1
	084	0-8053-1753-3	2
	085	0-8053-1753-3	2
	101	0-201-53771-0	1
	101	0-201-53771-0	1
	138	3-929821-31-1	1
	139	3-929821-31-1	1
	140	3-929821-31-1	1
	141	3-929821-31-1	1

**Das exists-Prädikat**

Allquantor simulieren

Beispiel: „Lehrstuhlbezeichnungen der Professoren, die **alle** von ihnen gelesenen Vorlesungen auch schon einmal geprüft haben.“

oder „Lehrstuhlbezeichnungen von Professoren, so dass keine von diesen gelesenen Vorlesung existiert, die von ihnen nicht geprüft wurde.“

```
select Lehrstuhlbezeichnung
from Professoren
where not exists
  (select *
   from Liest
   where Liest.PANr = Professoren.PANr
   and not exists (select *
    from Prüft
    where Prüft.PANr =
      Professoren.PANr
    and Prüft.V_Bezeichnung =
      Liest.V_Bezeichnung))
```

**Beispiel-Datenbank**

Professoren	PANr	Lehrstuhlbezeichnung	Stufe
	4711	Datenbank- und Informationssysteme	C4
	5588	Datenbanken und Informationssysteme	C4

Liest	PANr	V_Bezeichnung	Semester
	4711	Datenbanken I	WS 95/96
	4711	Theorie relationaler Datenbanken	WS 95/96
	4711	Datenbanken II	SS 96
	4711	Objektorientierte Datenbanken	SS 96
	5588	Datenbanken für Ingenieure	WS 95/96
	5588	Datenbanken I	WS 95/96
	5588	Verteilte Datenbanken	SS 96
	5588	Spezifikationsmethoden	SS 96

Prüft	PANr	Matrikelnummer	V_Bezeichnung	Note
	4711	HRO-912291	Datenbanken I	2.0
	4711	HRO-912291	Objektorientierte Datenbanken	2.3
	5588	MD-891372	Datenbanken I	1.3
	5588	MD-891372	Spezifikationsmethoden	2.7

**Bezug zum Tupelkalkül**

**exists-Prädikat:**  $\exists$ -Quantor des Tupelkalküls

andere Schachtelungsoperatoren ebenfalls auf Quantoren zurückführen

$\forall$ -Quantor mit  $\forall\phi \equiv \neg\exists\neg\phi$  simulieren

**SQL-92: Tupelbildungen**

**row constructors** bilden Tupel aus Konstanten oder Attributen  $(e_1, \dots, e_n)$

```
where ( select Studienfach, Immatrikulationsdatum
from Studenten
where Matrikelnummer = 'HRO-912291')
=
('Informatik', '1.10.91')
```

- Attribute müssen *kompatibel* sein (nächste Folie)
- Tupelvergleich

$$(a_1, \dots, a_n) < (b_1, \dots, b_n)$$

wahr, wenn ein  $j$  existiert, für das  $a_j < b_j$  und  $a_i = b_i$  für alle  $i < j$  gilt  
(lexikographische Ordnung)

**Kompatible Attribute**

Attribute sind kompatibel bei kompatiblen Wertebereichen

Zwei Wertebereiche sind **kompatibel**, wenn sie

- gleich sind oder
- beide auf `character` basierende Wertebereiche sind (unabhängig von der Länge der Strings) oder
- beide numerische Wertebereiche sind (unabhängig von dem genauen Typ) (wie `integer` oder `float`)

Kompatible Attribute können in Vergleichen und Mengenoperationen benutzt werden

**SQL-89: Vereinigung**

SQL-89: Vereinigung **union** einzige Mengenoperation

SFW\_block1 **union** SFW\_block2

Beispiel:

```
select A, B, C
from R1
union
select A, C, D
from R2
```

Attribute A von R1 und A von R2,  
B von R1 und C von R2  
C von R1 und D von R2  
müssen kompatibel sein

**SQL-89: Vereinigung**

Ergebnis: Attributnamen des linken Operanden

R1	A	B	C
	1	2	3
	2	3	4

R1	A	C	D
	2	2	3
	5	3	2

R1 union R2	A	B	C
	1	2	3
	2	3	4
	2	2	3
	5	3	2

Vereinigung nur als "äußerste" Operation erlaubt.

**Simulation der Differenz**

Beispiel: „Mitarbeiter, die keine Studenten sind“

$\pi$  [PANr] (Mitarbeiter) –  $\pi$  [PANr] (Studenten)

in SQL-89:

```
select PANr from Mitarbeiter
where PANr not in ( select PANr
                    from Studenten )
```

**Beispieldatenbank**

Mitarbeiter	PANr	AngNr	Fachbereich	Gehalt	Raum	Einstellung
	4711	HRO-	Informatik	6000	209	01.03.94
	5588	MD-5267	Informatik	6000	304	01.04.94
	6834	MD-	Mathemati	750	309	01.09.94
	7754	HRO-	Informatik	550	218	01.10.94
	8832	MD-4567	Informatik	2800	302	01.08.94
	9912	HRO-	Linguistik	2600	008	01.01.93

Studenten	PANr	Matrikelnummer	Studienfach	Immatrikulationsdatum
	6834	MD-891372	Informatik	01.10.89
	7754	HRO-912291	Informatik	01.10.91

**Vereinigung und äußere Verbunde**

Beispiel: „für jede Person Telefonnummer, falls vorhanden“

```
select *
from Personen left outer natural join Pers_Telefon;
```

simuliert durch **union** in SQL-89:

```
select P.PANr, P. Vorname, P.Nachname, P.PLZ,
       P.Ort, P.Strasse, P.HNr, P.Geburtsdatum, T. Telefon
from Personen P, Pers_Telefon T
where P.PANr = T. PANr
union
select P.PANr, P. Vorname, P.Nachname, P.PLZ,
       P.Ort, P.Strasse, P.HNr, P.Geburtsdatum, null
from Personen P
where not exists ( select *
                  from Pers_Telefon T
                  where P.PANr = T.PANr )
```

**Beispieldatenbank**

Personen	PANr	Vorname	Nachname	PLZ	Ort	Straße	HNr	Geb.datum
	4711	Andreas	Heuer	18209	DBR	BHS	15	31.10.1958
	5588	Gunter	Saake	39106	MD	STS	55	05.10.1960
	6834	Michael	Korn	39104	MD	BS	41	24.09.1974
	7754	Andreas	Möller	18209	DBR	RS	31	25.02.1976
	8832	Tamara	Jagellovsk	38106	BS	GS	12	11.11.1973
	9912	Antje	Hellhof	18059	HRO	AES	21	04.04.1970
	9999	Christa	Loeser	69121	HD	TS	38	10.05.1969

Pers_Telefon	PANr	Telefon
	4711	038203-12230
	4711	0381-498-3401
	4711	0381-498-3427
	5588	0391-345677
	5588	0391-5592-3800
	9999	06221-400177

**Vereinigung, Durchschnitt und Differenz in SQL-92**

**union, intersect** und **except** orthogonal in andere Anfragen einsetzbar

```
select count (*)
from ( (select PANr from Professoren)
      union
      (select PANr from Studenten) )
```

**corresponding**-Klausel: zwei Relationen nur über ihren gemeinsamen Bestandteilen (= gleiche Attributnamen) vereinigen

```
select count (*)
from ( Professoren union corresponding Studenten) )
```

**Vereinigung in SQL-92**

Beispiel:

R1	A	B	C
	1	2	3
	2	3	4

R2	A	C	D
	2	2	3
	5	3	2

R1 union corresponding R2	A	C
	1	3
	2	4
	2	2
	5	3

**Vergleich Relationenalgebra und SQL**

Relationenalgebra	SQL-89	SQL-92
Projektion	<b>select distinct</b>	<b>select distinct</b>
Selektion	<b>where</b> ohne Schachtelung	<b>where</b> ohne Schachtelung
Verbund	<b>from, where</b>	<b>from, where</b> <b>from</b> mit <b>join</b> oder <b>natural join</b>
Umbenennung	<b>from</b> mit Tupelvariable	<b>from</b> mit Tupelvariable <b>as</b>
Differenz	<b>where</b> mit Schachtelung	<b>where</b> mit Schachtelung <b>except corresponding</b>
Durchschnitt	<b>where</b> mit Schachtelung	<b>where</b> mit Schachtelung <b>intersect corresponding</b>
Vereinigung	<b>union</b> (nicht orthogonal)	<b>union corresponding</b>

**Weitere Sprachkonstrukte von SQL**

- Operationen auf Wertebereichen
- Aggregatfunktionen
- **group by** und **having**
- Quantoren und Mengenvergleiche
- Beispiele für Selbst-Verbund
- **order by**
- Nullwerte

**Operationen auf Wertebereichen**

innerhalb von **select** und **where**: statt Attribute auch *skalare Ausdrücke*

numerischen Wertebereiche: etwa +, -, x, /

Strings: **char\_length**, Konkatenation ||, **substring** (Teilzeichenkette suchen)

Datumstypen, Zeitintervalle: **current\_date**, **current\_time**, +, -, x

Ausdrücke werden tupelweise ausgewertet

```
select ISBN, Preis ÷ 1.44
from Buch_Versionen
```

ergibt

Ergebnis	ISBN	
	3-89319-175-5	54,86
	0-8053-1753-8	50,24
	0-8053-1753-8	61,70
	0-201-53771-0	60,73
	3-929821-31-1	54,86



- zweite Spalte im Ergebnis nicht benannt
- in SQL-89 über Spaltennummer identifizierbar:

```
select 2 from Ergebnis
```

in SQL-92: Attributname zuordnen:

```
select ISBN, Preis ÷ 1.44 as Dollar_Preis
from Buch_Versionen
```

ISBN	Dollar_Preis
3-89319-175-5	54,86
0-8053-1753-8	50,24
0-8053-1753-8	61,70
0-201-53771-0	60,73
3-929821-31-1	54,86



## Aggregatfunktion

oder built-in-Funktionen: tupelübergreifend

- **count**: Anzahl der Werte einer Spalte oder (Spezialfall **count(\*)**) Anzahl der Tupel einer Tabelle
- **sum**: Summe der Werte einer Spalte
- **avg**: arithmetisches Mittel der Werte einer Spalte
- **max** bzw. **min**: größter bzw. kleinster Wert einer Spalte

Argumente einer Aggregatfunktion:

- Attribut der durch **from** spezifizierten Relation
- gültiger skalarer Ausdruck
- bei **count** auch \*

Vor Argument (außer bei **count(\*)**) optional: **distinct** oder **all** (**all** Voreinstellung)

Nullwerte werden vor Anwendung aus Wertemenge eliminiert (außer bei **count(\*)**)



## Aggregatfunktion: Beispiele

```
select sum(Preis) from Buch_Versionen
```

406,65

```
select count(*)  
from Professoren
```

```
select count(distinct PANr)  
from Prüft
```

```
select avg (Note)  
from Prüft  
where V_Bezeichnung = `Datenbanken I`
```

„Studenten, die mindestens eine Prüfung mit einer Note gemacht haben, die besser als der Durchschnitt ist“

```
select Matrikelnummer  
from Prüft  
where Note < ( select avg (Note) from Prüft )
```

**group by and having**

```
select . . .  
from . . .  
[where . . .]  
[group by attributliste ]  
[having bedingung ]
```

Semantik (virtuelle geschachtelte Relation): Relationenschema  $R$  und Attributmeng  
hinter Gruppierung  $G$ , schachteln nach Attributen  $R - G$ , d.h. für gleiche  $G$ -Werte  
werden Resttupel in Relation gesammelt

**having** ist Selektionsbedingung auf gruppierter Relation: darf neben Gruppierungs-  
attributen auch auf beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen  
Bezug nehmen

**Beispielanfrage**

```
select A, sum (D)  
from R  
group by A, B  
having A<4 and sum(D) < 10 and max(C) = 4
```



**Auswertungsreihenfolge von group by and having**

Schritt 1:

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7

aus **from** und **where** resultierende Relation

Schritt 2:

A	B	N	
		C	D
1	2	3	4
		4	5
2	3	3	4
3	3	4	5
		6	7

**group by** A, B



**Auswertungsreihenfolge von group by and having**

Schritt 3:

A	sum(D)	N	
		C	D
1	9	3	4
		4	5
2	4	3	4
3	12	4	5
		6	7

**select** A, **sum(D)**

Schritt 4:

A	sum(D)
1	9

**having** A < 4 **and** sum(D) < 10 **and** max(C) = 4

**Weiteres Beispiel (auf der Datenbank 1)**

„Personen, die mehr als ein Buch ausgeliehen haben, mit der Anzahl der ausgeliehenen Bücher“

```
select Name, count (Bücher.ISBN)
from Bücher, Ausleihe
where Ausleihe.ISBN = Bücher.ISBN
group by Name
having count(Bücher.ISBN) > 1
```

wird in folgenden Schritten abgearbeitet:

- from** Bücher, Ausleihe  
**where** Ausleihe.ISBN = Bücher.ISBN

Bücher.ISBN	Titel	Verlag	Ausleihe.ISBN	Name
0-8053-1753-8	...	Benj. Cummings	0-8053-1753-8	Schmitz
0-8053-1753-8	...	Benj. Cummings	0-8053-1753-8	Derichsweiler
3-8244-2021-X	...	DUV	3-8244-2021-X	Radermacher
3-8244-2075-9	...	DUV	3-8244-2075-9	Radermacher



- group by** Name

Bücher.ISBN	Titel	Verlag	Ausleihe.ISBN	Name
0-8053-1753-8	...	Benj. Cummings	0-8053-1753-8	Schmitz
0-8053-1753-8	...	Benj. Cummings	0-8053-1753-8	Derichsweiler
3-8244-2021-X	...	DUV	3-8244-2021-X	Radermacher
3-8244-2075-9	...	DUV	3-8244-2075-9	

- having count** (Bücher.ISBN) > 1

Bücher.ISBN	Titel	Verlag	Ausleihe.ISBN	Name
3-8244-2021-X	...	DUV	3-8244-2021-X	Radermacher
3-8244-2075-9	...	DUV	3-8244-2075-9	

- select** Name, count(Bücher.ISBN)

Name	
Radermacher	2

**Weitere Beispiele (auf der Datenbank 2)**

„pro Person die Anzahl der von ihr ausgeliehenen Bücher“

```
select count(*) as Anzahl, PANr
from Ausleihe
group by PANr
```

Anzahl	PANr
2	7754
1	4711
1	5588
2	9912

„Personen, die mindestens 2 Bücher ausgeliehen haben“

```
select count(*), PANr
from Ausleihe
group by PANr
having count(*) > 1
```

	PANr
2	7754
2	9912

**Gruppierung: weitere Beispiele (auf Datenbank2)**

„Studenten, deren Durchschnittsnote besser ist als die Gesamtdurchschnittsnote aller Prüfungen“

```
select Matrikelnummer
from Prüft
group by Matrikelnummer
having avg(Note) < (select avg(Note) from Prüft)
```



## Quantoren und Mengenvergleiche

attribut  $\theta$  { **all** | **any** | **some** } ( **select** attribut  
**from** ... **where** ... )

$\theta$  Vergleichsoperator, **all** Allquantor, **any**, **some** Existenzquantoren

„Studenten, die schon einmal geprüft wurden“

```
select PANr, Immatrikulationsdatum  
from Studenten  
where Matrikelnummer = any ( select Matrikelnummer  
                                from Prüft )
```

„schlechteste Note des Studenten mit der Matrikelnummer ‚HRO-912291‘“

```
select Note  
from Prüft  
where Matrikelnummer = `HRO-912291`  
and Note  $\geq$  all ( select Note  
                    from Prüft  
                    where Matrikelnummer = `HRO-912291` )
```

Simulation der max-Funktion!



## Quantoren und Mengenvergleiche

Anwendbarkeit eingeschränkt: Test auf Mengen-Gleichheit

$$\forall x \in M_1 : \exists x \in M_2 \wedge \forall x \in M_2 : \exists x \in M_1$$

in SQL so nicht umsetzbar

Beispiel:

„Gib alle Bücher aus, an denen ‚Vossen‘ und ‚Witt‘ gemeinsam als Autoren beteiligt waren“

**Selbst-Verbund**

letzte Anfrage erst mit Selbst-Verbund zu lösen  
Vergleich von Wertemengen

```
select B_A_1.ISBN
from Buch_Autor B_A_1, Buch_Autor B_A_2
where B_A_1.ISBN = B_A_2.ISBN
      and B_A_1.Autor = 'Vossen' and B_A_2.Autor = 'Witt'
```

Zwischenergebnis:

B_A_1.ISBN	B_A_1.Autor	B_A_2.ISBN	B_A_2.Autor
3-89319-175-5	Vossen	3-89319-175-5	Vossen
3-89319-175-5	Vossen	3-89319-175-5	Witt
3-89319-175-5	Vossen	0-8053-1753-8	Elmasri
...	...	...	...
3-89319-175-5		3-89319-175-5	
...	...	...	...

**order by-Klausel**

Menge von Tupeln  $\rightsquigarrow$  Liste  
**order by** attributliste

Beispiel:

```
select Matrikelnummer, Note
from Prüft
where V_Bezeichnung = 'Datenbanken I'
order by Note asc
```

aufsteigend (**asc**) oder absteigend (**desc**) sortieren Sortierung wird auf das Ergebnis der jeweils vorangehenden SFW-Anfrage angewendet, also FALSCH:

```
select Matrikelnummer
from Prüft
where V_Bezeichnung = 'Datenbanken I'
order by Note (falsch, weil Note nicht in der Ausgabeliste steht!)
```

**Behandlung von Nullwerten**

- skalare Ausdrücke: Ergebnis **null**, sobald Nullwert in die Berechnung eingeht
- In allen Aggregatfunktionen bis auf **count(\*)** werden Nullwerte vor Anwendung der Funktion entfernt
- Fast alle Vergleiche mit Nullwert ergeben Wahrheitswert **unknown** (statt **true** oder **false**). Ausnahme: **is null** ergibt **true**, **is not null** ergibt **false**
- Boolesche Ausdrücke basieren dann auf dreiwertiger Logik

and	true	unknown	false	or	true	unknown	false
true	true	unknown	false	true	true	true	true
unknown	unknown	unknown	false	unknown	true	unknown	unknown
false	false	false	false	false	true	unknown	false

  

not	
true	false
unknown	unknown
false	true

**5.3. SQL - Änderungsoperationen**

- Einfügen von Tupeln in Basisrelationen: **insert**
- Löschen von Tupeln aus Basisrelationen: **delete**
- Ändern von Tupeln in Basisrelationen: **update**

Diese Operationen jeweils als

- Eintupel-Operationen (etwa die Erfassung einer neuen Ausleihung)
- Mehrtupel-Operationen (erhöhe das Gehalt aller Mitarbeiter um 4.5%)

SQL: vor allem Mehrtupel-Operationen

**update**

**update** basisrelation  
**set** attribut\_1 = ausdruck\_1  
...  
Attribut\_n = ausdruck\_n  
[ **where** bedingung ]

Name	Gehalt
Meyer	3000
Schulz	3500
Bond	7200
Schulz	4400

**update** Angestellte  
**set** Gehalt = Gehalt + 1000  
**where** Gehalt < 5000

Name	Gehalt
Meyer	4000
Schulz	4500
Bond	7200
Schulz	5400

**update** Angestellte  
**set** Gehalt = 6000  
**where** Name = 'Bond'

**update** Angestellte  
**set** Gehalt = 3000

**delete**

**delete**  
**from** basisrelation  
[ **where** bedingung ]

**delete from** Ausleihe  
**where** Invnr = 4711

Standardfall ist Löschen mehrerer Tupel:

**delete from** Ausleihe  
**where** PANr = 7754

Löschen der gesamten Relation:

**delete from** Ausleihe

**insert**

```
insert  
into basisrelation [ (attribut_1, ..., attribut_n) ]  
values (konstante-1, ..., konstante_n)
```

```
insert  
into Buch (Invr, Titel)  
values (4867, 'Wissensbanken')
```

```
insert  
into Buch  
values (4867, 'Wissensbanken', '3-876', 'Karajan')
```

```
insert  
into basisrelation [ (attribut_1, ..., attribut_n) ]  
      SQL-anfrage
```

```
insert into Kunde  
      ( select LName, LAdr, 0  
        from Lieferant )
```