



Informations- und Wissensmanagement

Kapitel 4: **Relationale Datenbanksprachen**

Join (Natural Join) (1)

[Einleitung](#)

[SQL-Kern](#)

[Weitere Konstrukte](#)

[Änderungen](#)

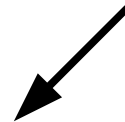
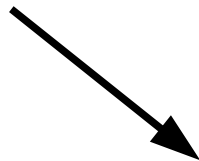
[Schluß](#)

Ausleihe

Invnr	Name
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

Bücher

Invnr	...	Autor
0007	...	Fleming
1201	...	Heuer
4711	...	Vossen
4712	...	Ullman
4717	...	Wirth



Invnr	Name	...	Autor
0007	Müller	...	Fleming
1201	Schulz	...	Heuer
4711	Meyer	...	Vossen
4712	Meyer	...	Ullman



Join (Natural Join) (2)

[Einleitung](#)

[SQL-Kern](#)

[Weitere Konstrukte](#)

[Änderungen](#)

[Schluß](#)

Ausleih

Invnr	ISBN	Name
4711	1234	Meyer
1201	5678	Schulz
0007	9012	Müller
4712	3456	Meyer

Bücher

Invnr	ISBN	Autor
0007	9012	Fleming
1201	5678	Heuer
4711	1234	Vossen
4712	4713	Ullman
4717	1234	Wirth

- Was ist das Join-Ergebnis?
 - Welches Schema?
 - Welche Tupel?





Join (Natural Join) (3)

[Einleitung](#)

[SQL-Kern](#)

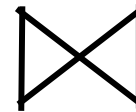
[Weitere Konstrukte](#)

[Änderungen](#)

[Schluß](#)

Ausleihe

Invnr	Name
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer



Bücher

ISBN	Autor
9012	Fleming
5678	Heuer
1234	Vossen
4713	Ullman
1234	Wirth

Invnr	Name	ISBN	Autor
4711	Meyer	9012	Fleming
4711	Meyer	5678	Heuer
4711	Meyer	1234	Vossen
...
1201	Schulz	9012	Fleming
1201	Schulz	5678	Heuer
...

Kartesisches Produkt





Aggregatfunktionen (1)

[Einleitung](#)

[SQL-Kern](#)

[Weitere
Konstrukte](#)

[Änderungen](#)

[Schluß](#)

- Prinzip: Berechnung eines Werts aus allen (oder zumindest) Werten eines Attributs.
- Aggregatfunktionen in Standard SQL: COUNT(), SUM(), MIN(), MAX(), AVG()
- SQL-Erweiterungen beinhalten zusätzliche Aggregatfunktionen (Statistik, Physik).



Aggregatfunktionen (2)

Ausleih

Invnr	ISBN	Name
4711	1234	Meyer
1201	5678	Schulz
0007	9012	Müller
4712	3456	Meyer

- `count(Invnr)=4,`
- `count(Name)=3`

Group-by Operator (1)

Marke	Datum	Bundesland	Anzahl
BMW	07.01.94	Hessen	28
BMW	08.01.94	Bayern	37
BMW	07.01.94	Saarland	41
Opel	07.01.94	Hessen	48
Opel	08.01.94	Bayern	62
Opel	08.01.94	Saarland	5
Opel	09.01.94	Saarland	95
Audi	07.01.94	Hessen	55
Audi	08.01.94	Bayern	52
Audi	09.01.94	Bayern	27
Audi	10.01.94	Bayern	62

Marke
→
sum

Marke	Anzahl
BMW	106
Opel	210
Audi	196

Marke
→
avg

Marke	Anzahl
BMW	35,33
Opel	52,5
Audi	49

Marke
→
max

Marke	Anzahl
BMW	41
Opel	95
Audi	62

Bundesland
→
sum

Bundesland	Anzahl
Hessen	131
Bayern	240
Saarland	141

Group-by Operator (2)

Marke	Datum	Bundesland	Anzahl
BMW	07.01.94	Hessen	28
BMW	08.01.94	Bayern	37
BMW	07.01.94	Saarland	41
Opel	07.01.94	Hessen	48
Opel	08.01.94	Bayern	62
Opel	08.01.94	Saarland	5
Opel	09.01.94	Saarland	95
Audi	07.01.94	Hessen	55
Audi	08.01.94	Bayern	52
Audi	09.01.94	Bayern	27
Audi	10.01.94	Bayern	62

Marke,
Bundes-
land
→
sum

Marke	Bundesland	Anzahl
BMW	Hessen	28
BMW	Bayern	37
BMW	Saarland	41
Opel	Hessen	48
Opel	Bayern	62
Opel	Saarland	100
Audi	Hessen	55
Audi	Bayern	141



Relationale Datenbanksprachen

[Einleitung](#)

[SQL-Kern](#)

[Weitere
Konstrukte](#)

[Änderungen](#)

[Schluß](#)

- SQL-Kern,
- Weitere Sprachkonstrukte von SQL,
- SQL-Versionen.





SQL-Kern

Einleitung

SQL-Kern

- [Übersicht](#)

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- **select**
 - Projektion, ggf. Umbenennungen von Attributen
 - arithmetische Operationen, Aggregatfunktionen
- **from**
 - Relationen, ggf. Umbenennungen
- **where**
 - Selektions-, Verbundbedingungen,
 - geschachtelte Anfragen (wieder SFW-Block),
- **group by**
 - Gruppierung für Aggregatfunktionen,
- **having**
 - Selektionsbedingungen an Gruppen.



Laufendes Beispiel

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

Ausleih

INV.NR	NAME
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

Bücher

INV.NR	TITEL	ISBN	AUTOR
0007	Dr. No	3-324	Fleming
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-345	Vossen
4712	Datenbanken	3-345	Ullman
4717	PASCAL	3-989	Wirth

from-Klausel

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Syntax:
`select * from relationenliste`
- Beispiel:
`select * from Bücher`
liefert die gesamte Relation Bücher.

Bücher

INV.NR	TITEL	ISBN	AUTOR
0007	Dr. No	3-324	Fleming
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-345	Vossen
4712	Datenbanken	3-345	Ullman
4717	PASCAL	3-989	Wirth



Kartesisches Produkt

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Bei mehr als einer Relation hinter **from**: Kartesisches Produkt.

```
select * from Bücher, Ausleih
```

Invnr	Name	ISBN	Autor
4711	Meyer	9012	Fleming
4711	Meyer	5678	Heuer
4711	Meyer	1234	Vossen
...
1201	Schulz	9012	Fleming
1201	Schulz	5678	Heuer
...



Self-Join – Illustration (1)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- **Self-Join:** `select * from Person, Person`
 - Kartesisches Produkt einer Relation **mit sich selbst.**
- Zugrundeliegende Relation:

Person

Vorname	Name
Erik	Buchmann
Klemens	Böhm

Attributnamen kommen mehrmals vor!

- Gewünscht: Alle Personen-Paare:

Vorname	Name	Vorname	Name
Erik	Buchmann	Erik	Buchmann
Klemens	Böhm	Erik	Buchmann
Erik	Buchmann	Klemens	Böhm
Klemens	Böhm	Klemens	Böhm

Self-Join – Illustration (2)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- Einführung von Tupelvariablen:
Etwa auf eine Relation mehrfach zugreifen.
select * **from** Person *eins*, Person *zwei*
where *eins*.Alter < *zwei*.Alter
- Ergebnis hat vier Spalten:
eins.Name, *eins*.Vorname,
zwei.Name, *zwei*.Vorname
- *Selbst-Verbund* (Self-Join)
für tupelübergreifende Selektionen.
- Weiteres Beispiel:
select * **from** Person *eins*, Person *zwei*
where *eins*.Alter < *zwei*.Alter

Self-Join – Illustration (3)

```
mysql> select * from Person eins, Person zwei;
```

Vorname	Name	Vorname	Name
Markus	Bestehorn	Markus	Bestehorn
Erik	Buchmann	Markus	Bestehorn
Klemens	Böhm	Markus	Bestehorn
Mirco	Stern	Markus	Bestehorn
Markus	Bestehorn	Erik	Buchmann
Erik	Buchmann	Erik	Buchmann
Klemens	Böhm	Erik	Buchmann
Mirco	Stern	Erik	Buchmann
Markus	Bestehorn	Klemens	Böhm
Erik	Buchmann	Klemens	Böhm
Klemens	Böhm	Klemens	Böhm
Mirco	Stern	Klemens	Böhm
Markus	Bestehorn	Mirco	Stern
Erik	Buchmann	Mirco	Stern
Klemens	Böhm	Mirco	Stern
Mirco	Stern	Mirco	Stern

```
16 rows in set (0.00 sec)
```

```
mysql>
```

SQL-92: spezielle Aspekte

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Verbunde als explizite Operatoren:
- Kartesisches Produkt:
select * from Bücher, Ausleih
select * from Bücher **cross join** Ausleih
- Verbund über Verbundbedingungen:
select * from Bücher, Ausleih
where Bücher.Invnr = Ausleih.Invnr
- **join-Operator: θ -Verbund**
select *
from Bücher **join** Ausleih
on Bücher.Invnr = Ausleih.Invnr

Theta steht für beliebige Operatoren



Theta Join – Beispiel

- Einleitung
- SQL-Kern
 - Übersicht
 - [from](#)
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

Prof

PName	PVorname
Saake	Gunter
Rautenstrauch	Claus
Böhm	Klemens

DKE

Vorname	Name
Klemens	Böhm
Erik	Buchmann
Ralf	Duckstein



PName	PVorname	Vorname	Nachname
Saake	Gunter	Klemens	Böhm
Saake	Gunter	Ralf	Duckstein
Rautenstrauch	Claus	Klemens	Böhm
Rautenstrauch	Claus	Erik	Buchmann
Rautenstrauch	Claus	Ralf	Duckstein
Böhm	Klemens	Ralf	Duckstein

Statt < auch ≥, ≠, ≈ usw. möglich.





Weitere Verbunde in SQL-92

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- Gleichverbund:

```
select * from Bücher join Ausleih  
      using (Inventarnr)
```
- natürlicher Verbund (doppelte Attribute eliminiert):

```
select * from Bücher natural join  
Ausleih
```

Besser als herkömmliche Formulierung, weil

 - übersichtlicher,
 - weniger fehleranfällig
(z.B. vergißt man leicht ein Attribut).
- Jeder SFW-Block kann in SQL-92 auch hinter **from** Klausel eingesetzt werden (Orthogonalität). **Nicht mit SQL-89.**



Äußere Verbunde (1)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

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.
- Beispiel:

```
select * from Verlage right outer join  
Bücher2 using (VName)
```

Basisrelationen

```
X aterm
mysql> select * from Buecher;
+-----+-----+-----+-----+
| InvNr | Titel          | ISBN  | Autor  |
+-----+-----+-----+-----+
| 7     | Dr. No         | 3-324 | Fleming|
| 1201  | Objektbanken  | 3-111 | Heuer  |
| 4711  | Datenbanken   | 3-345 | Vossen |
| 4712  | Datenbanken   | 3-345 | Ullman |
| 4717  | PASCAL         | 3-989 | Wirth  |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from Ausleih;
+-----+-----+
| InvNr | Name  |
+-----+-----+
| 7     | Müller|
| 1201  | Schulz|
| 4711  | Meyer |
| 4712  | Meyer |
+-----+-----+
4 rows in set (0.00 sec)

mysql> █
```

Right- und Left Outer Join

```
X aterm
mysql> select Titel, Name from Buecher right outer join Ausleih using (InvNr);
+-----+-----+
| Titel      | Name  |
+-----+-----+
| Dr. No     | Müller|
| Objektbanken | Schulz|
| Datenbanken | Meyer |
| Datenbanken | Meyer |
+-----+-----+
4 rows in set (0.00 sec)

mysql> select Titel, Name from Buecher left outer join Ausleih using (InvNr);
+-----+-----+
| Titel      | Name  |
+-----+-----+
| Dr. No     | Müller|
| Objektbanken | Schulz|
| Datenbanken | Meyer |
| Datenbanken | Meyer |
| PASCAL     | NULL  |
+-----+-----+
5 rows in set (0.00 sec)

mysql> █
```

Äußere Verbunde (2)

- Einleitung
- SQL-Kern
- Übersicht
- [from](#)
- select
- where
- Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

LinkeRelation		RechteRelation	
A	B	B	C
1	2	3	4
2	3	4	5

Natural Join		
A	B	C
2	3	4

Outer Join		
A	B	C
1	2	NULL
2	3	4
NULL	4	5

Left Outer Join		
A	B	C
1	2	NULL
2	3	4

Right Outer Join		
A	B	C
2	3	4
NULL	4	5

Die select-Klausel

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- Abschließende Projektion, Zielliste.
- Syntax:

```
select [distinct] { attribut |  
                    arithmetischer-ausdruck |  
                    aggregat-funktion }  
from ...
```
- Bestandteile:
 - Attribute aus **from**-Relationen,
 - arithmetische Ausdrücke
über Attributen und Konstanten,
 - Aggregatfunktionen über Attributen.
- **distinct**: Ergebnismenge statt Multimenge
→ Duplikateliminierung



Laufendes Beispiel

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

Ausleih

INV.NR	NAME
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

Bücher

INV.NR	TITEL	ISBN	AUTOR
0007	Dr. No	3-324	Fleming
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-345	Vossen
4712	Datenbanken	3-345	Ullman
4717	PASCAL	3-989	Wirth





Projektion: Menge oder Multimenge

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- **select** Name **from** Ausleih

NAME

Meyer

Schulz

Müller

Meyer

- **select distinct** Name **from** Ausleih

NAME

Meyer

Schulz

Müller



Tupelvariablen und Relationennamen (1)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- Attributnamen durch Präfix ergänzen:

```
select ISBN from Bücher
```

und

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

- Tupelvariable kann benutzt werden:

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

Tupelvariablen und Relationennamen (2)

aus welcher
Relation stammt
ISBN?

```
select ISBN, Titel, Stichwort falsch  
from Bücher, Buch_Stichwort  
where Bücher.ISBN = Buch_Stichwort.ISBN
```

```
select Bücher.ISBN, Titel, Stichwort richtig  
from Bücher, Buch_Stichwort  
where Bücher.ISBN = Buch_Stichwort.ISBN
```

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß



Die where-Klausel

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Selektionsbedingung oder Verbundbedingung.
- Syntax:
`select ...from ...where bedingung`
- Bedingung:
 - *Konstanten-Selektion*:
`attribut θ konstante`
 - *Attribut-Selektion* zwischen Attributen mit kompatiblen Wertebereichen:
`attribut1 θ attribut2`





Konstantenselektion

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Beispiel:
select *
from AUSLEIH
where NAME = 'Meyer'

Ausleih

INV.NR	NAME
4711	Meyer
4712	Meyer



Verbundbedingung

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- `relation1.attribut = relation2.attribut`
- **Beispiel: natürlicher Verbund**

```
select Bücher.Titel,  
        Bücher_Stichwort.Stichwort  
from Bücher, Buch_Stichwort  
where Bücher.ISBN =  
        Buch_Stichwort.ISBN
```
- Auch Gleichverbund und θ -Verbund erlaubt.
(*Gleichverbund* – Attribute, die nicht gleich heißen, mit ‚==‘ vergleichen.)



Bereichsselektion

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

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

- Abkürzung für

attribut \geq konstante1 **and**

attribut \leq konstante2

- Beispiel:

```
select Matrikelnummer from Prüft  
where Note between 1.0 and 2.0
```



Ungewißheitsselektion (1)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- Theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung.
- Syntax:
`attribut like spezialkonstante`
- Spezialkonstante kann beinhalten
 - '%' – kein oder beliebig viele Zeichen,
 - '_' – genau ein Zeichen.





Ungewißheitsselektion (2)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- 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 =
    'BenjXFCummingsSCHlumpf'
or ...
```





Weitere Bedingungen

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- *Null-Selektion*
`attribut is null`
- boolesche Ausdrücke mit Konnektoren **or**, **and** und **not**,
- *quantifizierte Bedingungen*,
wenn ein Argument in Vergleich Menge liefert
(**all**, **any**, **some** und **exists**;
wird gleich besprochen).





Schachtelung von Anfragen

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- **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 (1)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- **Syntax:**

```
attribut in ( SFW-block )
```

- **Beispiel:**

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

Bücher

TITEL	ISBN	AUTOR
Dr. No	3-324	Fleming
Objektbanken	3-111	Heuer
Datenbanken	3-345	Vossen
Datenbanken	3-345	Ullman
PASCAL	3-989	Wirth

Empfiehl

ISBN	PANr
3-111	7743
3-324	3234
3-345	5643
3-345	4563



Das in-Prädikat (2)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Abarbeitung:
 1. Ergebnis der inneren **select**-Anweisung hinter **in** als Liste von Konstanten einsetzen.
 2. Dann modifizierte Anfrage abarbeiten:

```
select Titel from Bücher
where ISBN in
('3-929821-31-1', '0-201-53771-0',
 '3-89319-175-5', '0-8053-1753-8')
```

- Lässt sich für diese Anfrage

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

eine äquivalente Anfrage ohne Schachtelung finden?



Verzahnt geschachtelte Anfragen (1)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- *Verzahnt geschachtelt* – in der inneren Anfrage Relationen- oder Tupelvariablen-Name aus dem **from**-Teil der äußeren Anfrage verwenden.
- Beispiel:

Personen

PANr	Name
...	...

Prüft

PANr	Matrikel	Vorlesung	Note
...

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



Verzahnt geschachtelte Anfragen (2)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- Abarbeitung

1. In der äußeren Anfrage
das erste `Personen`-Tupel untersuchen.
Ergebnis in innere Anfrage einsetzen.

2. Innere Anfrage

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

auswerten, liefert Werteliste (2.0, 2.3).



Verzahnt geschachtelte Anfragen (3)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- Abarbeitung (Fortsetzung):
 - Ergebnis der inneren Anfrage in die äußere einsetzen.
1.0 in (2.0, 2.3) ergibt **false**.
Ersten Prüfer nicht berücksichtigen.
 - In der äußeren Anfrage das zweite Personen-Tupel untersuchen usw.
- Was leistet Anfrage also, anschaulich gesprochen?



Verzahnt geschachtelte Anfragen (4)

Personen

PANr	Name
...	...

Prüft

PANr	Matrikel	Vorlesung	Note
...

- **select** Name
 from Personen
 where 1.0 **in** (**select** Note
 from Prüft
 where PANr = Personen.PANr)
- Wie sieht eine äquivalente Anfrage ohne Schachtelung aus?

Das exists-Prädikat (1)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Testet ob Ergebnis der inneren Anfrage nicht leer

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

Buch_Exemplar

Invnr	ISBN	Standort
...

Ausleihe

Invnr	Ausleiher-Nr	Ausleih-Datum	Frist
...

- Was leistet die Anfrage?
Als nicht geschachtelte Anfrage darstellbar?



Das exists-Prädikat (2)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Formulieren Sie die Anfrage

„Selektiere alle Bücher,
von denen alle Exemplare präsent sind.“



exists: Simulation des Allquantors (1)

- Einleitung
- SQL-Kern
 - Übersicht
 - from
 - select
 - where
 - Mengenop.
- Weitere Konstrukte
- Änderungen
- Schluß

- Beispiel – drei zugrundeliegende Relationen:

prüft

PANr	V_Bezeichnung
...	...

Professoren

PANr	Lehrstuhl
...	...

liest

PANr	V_Bezeichnung
...	...

- Gesucht: „Alle Professoren, die für alle Vorlesungen, die sie lesen, eine Prüfung abnehmen.“

exists: Simulation des Allquantors (2)

- Beispiel:

```
select Lehrstuhl  
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) )
```

Alle Vorlesungen,
die ein bestimmter Prof.
hält, aber nicht prüft.



Kompatible Attribute (1)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

- Attribute sind kompatibel bei kompatiblen Wertebereichen.
- Zwei Wertebereiche sind kompatibel, wenn sie
 - gleich sind,
 - beide auf `character` basierende Wertebereiche sind (unabhängig von der Länge der Strings), oder
 - beide numerische Wertebereiche sind (unabhängig vom genauen Typ) wie `integer` oder `float`.





Kompatible Attribute (2)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- Kompatible Attribute können in Vergleichen und Mengenoperationen benutzt werden.
- Beispiel: '`... where Salary < 50000`', und `Salary` ist vom Typ `float`.



SQL-92: Tupelbildungen (1)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- *row constructors* bilden Tupel aus Konstanten oder Attributen (e_1, \dots, e_n)
where (**select** Studienfach, Immadatum
from Studenten
where Matrikelnummer = 'HRO-912291')
=
('Informatik', '1.10.91')





SQL-92: Tupelbildungen (2)

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere
Konstrukte

Änderungen

Schluß

- $(a_1, \dots, a_n) < (b_1, \dots, b_n)$
 - wahr, wenn ein j existiert, für das $a_j < b_j$ und $a_i = b_i$ für alle $i < j$ gilt (lexikographische Ordnung).
Beispiel: (Böhm, Klemens, Magdeburg) < (Böhm, Klemens, Zürich)
- Attribute müssen *kompatibel* sein.
Beispiel: (Böhm, Klemens, 39114)
nicht vergleichbar mit (Böhm, Klemens, Zürich)

SQL-89: Vereinigung

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- Mengenop.

Weitere

Konstrukte

Änderungen

Schluß

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

```
SFW_block1 union SFW_block2
```

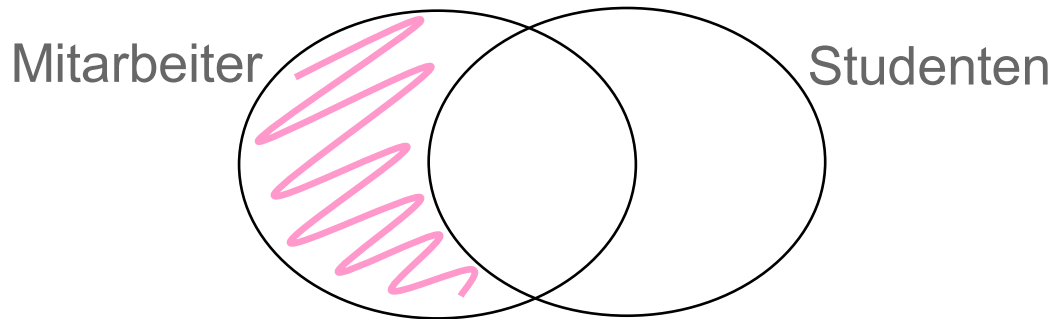
- *Positionsweise Vereinigung.*
- Beispiel:

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

Attributkompatibilität: A von R1 und A von R2,
B von R1 und C von R2, C von R1 und D von R2.

- Ergebnis: Attributnamen des linken Operanden.
- Vereinigung nur als „äußerste“ Operation erlaubt.

Simulation der Differenz



- In SQL:

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

- Gibt es hierzu äquivalente,
nicht geschachtelte Anfrage?



Vereinigung und äußere Verbunde

Einleitung

SQL-Kern

- Übersicht

- from

- select

- where

- [Mengenop.](#)

Weitere
Konstrukte

Änderungen

Schluß

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

umgesetzt

```
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 )
```

Vereinigung, Durchschnitt und Differenz

Duplikat-
eliminierung!

- SQL-92: **union**, **intersect** und **except** (Oracle: **minus**)
orthogonal in andere Anfragen einsetzbar.

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

- **corresponding**-Klausel: Zwei Relationen nur über ihre gemeinsamen Attribute vereinigen.

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

- D. h. alle anderen Bestandteile fallen weg.



Weitere Sprachkonstrukte von SQL

Einleitung

SQL-Kern

Weitere
Konstrukte

- [Übersicht](#)

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

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



Operationen auf Wertebereichen (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- Innerhalb von **select** und **where**:
Statt Attributen auch *skalare Ausdrücke*.
 - Numerische Wertebereiche: etwa + , - , * , /,
 - Strings: **char_length**, Konkatenation .. , **substring** (Teilzeichenkette),
 - Datumstypen, Zeitintervalle: **current_date**, **current_time**, + , - , *.

Operationen auf Wertebereichen (2)

- Ausdrücke werden tupelweise ausgewertet.

```
select ISBN, Preis / 1.16  
from Bücher
```

Ergebnis	ISBN	Preis
	3-324	45,23
	3-111	63,24
	3-345	33,34
	3-345	53,33
	3-989	18,99

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Opera-
tionen

- Aggre-
gation

- Gruppie-
rung

- Quantoren/
Selbst-
verbund

- order-by

- Nullwerte

Änderungen

Schluß



SQL-92-Spezialitäten: Umbenennung

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Opera-
tionen

- Aggre-
gation

- Gruppie-
rung

- Quantoren/
Selbst-
verbund

- order-by

- Nullwerte

Änderungen

Schluß

- Im vorigen Beispiel: Zweite Spalte nicht benannt.
- In SQL-92: Attributnamen zuordnen.

```
select ISBN, Preis / 1.16 as Netto  
from Bücher
```

Ergebnis	ISBN	Netto
	3-324	45,23
	3-111	63,24
	3-345	33,34
	3-345	53,33
	3-989	18,99





Aggregatfunktionen: Beispiele

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- Quantoren/Selbstverbund
- order-by
- Nullwerte
- Änderungen
- Schluß

- **select sum(all Preis) from Bücher**
- **select sum(distinct Preis) from Bücher**
- **select sum(Preis) from Bücher**
- **select count(*) from Professoren**
- **select count(distinct PANr) from Prüft**
- **select avg(all Note) from Prüft**
where V_Bezeichnung = 'Datenbanken I'
- **select Matrikelnummer from Prüft**
where Note < (select avg(all Note)
from Prüft)





Aggregatfunktionen (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- Built-in Funktionen: Tupelübergreifend.
 - **count**: Anzahl der Werte einer Spalte
 - **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 * für alle Tupel der Relation





Aggregatfunktionen (2)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Vor Argument (außer bei **count(*)**) optional: **distinct** oder **all** (**all** ist Voreinstellung).
- Nullwerte werden vor Anwendung aus Wertemenge eliminiert, außer bei **count(*)**.



Group-by Operator (1)

Marke	Datum	Bundesland	Anzahl
BMW	07.01.94	Hessen	28
BMW	08.01.94	Bayern	37
BMW	07.01.94	Saarland	41
Opel	07.01.94	Hessen	48
Opel	08.01.94	Bayern	62
Opel	08.01.94	Saarland	5
Opel	09.01.94	Saarland	95
Audi	07.01.94	Hessen	55
Audi	08.01.94	Bayern	52
Audi	09.01.94	Bayern	27
Audi	10.01.94	Bayern	62

Marke
→
sum

Marke	Anzahl
BMW	106
Opel	210
Audi	196

Marke
→
avg

Marke	Anzahl
BMW	35,33
Opel	52,5
Audi	49

Marke
→
max

Marke	Anzahl
BMW	41
Opel	95
Audi	62

Bundesland
→
sum

Bundesland	Anzahl
Hessen	131
Bayern	240
Saarland	141

Group-by Operator (2)

Marke	Datum	Bundesland	Anzahl
BMW	07.01.94	Hessen	28
BMW	08.01.94	Bayern	37
BMW	07.01.94	Saarland	41
Opel	07.01.94	Hessen	48
Opel	08.01.94	Bayern	62
Opel	08.01.94	Saarland	5
Opel	09.01.94	Saarland	95
Audi	07.01.94	Hessen	55
Audi	08.01.94	Bayern	52
Audi	09.01.94	Bayern	27
Audi	10.01.94	Bayern	62

Marke,
Bundes-
land
→
sum

Marke	Bundesland	Anzahl
BMW	Hessen	28
BMW	Bayern	37
BMW	Saarland	41
Opel	Hessen	48
Opel	Bayern	62
Opel	Saarland	100
Audi	Hessen	55
Audi	Bayern	141



group by und having (1)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Syntax

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

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





group by und having (2)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Beispiele von eben:
 - **select** Marke, sum(Anzahl)
from Zulassungen
group by Marke
 - **select** Marke, avg(all Anzahl)
from Zulassungen
group by Marke
 - **select** Marke, max(Anzahl)
from Zulassungen
group by Marke
 - **select** Bundesland, sum(Anzahl)
from Zulassungen
group by Bundesland





group by und having (3)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Beispiele von eben (Forts.):
 - **select** Marke, Bundesland, sum(Anzahl)
from Zulassungen
group by Marke, Bundesland
- Ist diese Anfrage auch zulässig?
 - **select** Marke, **Datum**, sum(Anzahl)
from Zulassungen
group by Marke, Bundesland





group by und having (5)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- **having** ist Selektionsbedingung auf gruppierter Relation,
- darf Bezug nehmen auf
 - Gruppierungsattribute,
 - beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen.
- Beispiel:
select Bundesland, count(*)
from Zulassungen
group by Bundesland
having sum(Anzahl) > 100





Gruppierung: Schema

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Relation REL:

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

- Anfrage:

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





Gruppierung: Schritt 1

Einleitung

SQL-Kern

Weitere Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

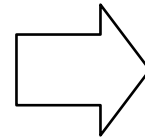
- Nullwerte

Änderungen

Schluß

- **from REL where B<4**

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



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



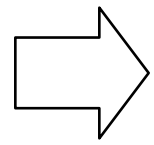


Gruppierung: Schritt 2

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- **group by A, B**

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



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



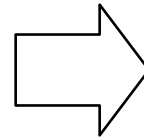


Gruppierung: Schritt 3

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

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

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



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

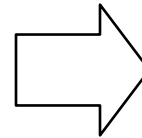




Gruppierung: Schritt 4

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

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



A	SUM(D)
1	9

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Opera-
tionen

- Aggre-
gation

- Gruppie-
rung

- Quantoren/
Selbst-
verbund

- order-by

- Nullwerte

Änderungen

Schluß





Gruppierung: Beispiele (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- Quantoren/Selbstverbund
- order-by
- Nullwerte
- Änderungen
- Schluß

- Basisrelation: Ausleihe

PANr	INV.NR
7754	4711
4711	1201
5588	0007
9912	4712
7754	2354
9912	6324

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

Anzahl	PANr
2	7754
1	4711
1	5588
2	9912





Gruppierung: Beispiele (2)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- Quantoren/Selbstverbund
- order-by
- Nullwerte
- Änderungen
- Schluß

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

- Ergebnis:

Anzahl	PANr
2	7754
2	9912

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

- Zugrundeliegende Relation:

Matrikelnummer	Note
...	...





Quantoren und Mengenvergleiche (1)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbst-
verbund

- order-by

- Nullwerte

Änderungen

Schluß

- Syntax

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

- Bedeutung: **all** Allquantor,
any, **some** Existenzquantoren, äquivalent.

- Beispiele:

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

„= any“
entspricht „in“

- Äquivalente Formulierung?
- Anfrage mit **any**, die nicht ohne weiteres mit **in** formulierbar?





Quantoren und Mengenvergleiche (2)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- **select** Note **from** Prüft
where Matrikelnummer = 'HRO-912291'
and Note \geq **all** (
 select Note **from** Prüft
 where Matrikelnummer = 'HRO-912291')
- Wie sieht eine äquivalente Anfrage aus, die ohne Allquantor und Schachtelung auskommt?





Quantoren und Mengenvergleiche (3)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- [Quantoren/Selbstverbund](#)
- order-by
- Nullwerte
- Änderungen
- Schluß

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

- Anwendbarkeit eingeschränkt:
Test auf Mengen-Gleichheit geht nicht.
Definition von 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:
*Gib alle Bücher aus, an denen
'Heuer und 'Saake' gemeinsam
als Autoren beteiligt waren.*

Bücher

ISBN	AUTOR
3-324	Fleming
3-111	Heuer
3-111	Saake





Selbst-Verbund (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbst-verbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- Letzte Anfrage erst mit Selbst-Verbund zu lösen.
- Vergleich von Wertemengen:

```
select BA_1.ISBN, BA_1.Autor,  
        BA_2.Autor  
from Bücher BA_1, Bücher BA_2  
where BA_1.ISBN = BA_2.ISBN  
and BA_1.Autor = 'Heuer'  
and BA_2.Autor = 'Saake'
```

ISBN	AUTOR	AUTOR
3-111	Heuer	Saake





Selbst-Verbund (2)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbst-verbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- „Alle Prüfer, die zwei oder mehr Studenten geprüft haben.“
- Zählen von Wertemengen

Prüft

PANr	Matrikelnummer	Note
...

```
select distinct X.PANr
from Prüft X, Prüft Y
where X.PANr = Y.PANr
and X.Matrikelnummer <>
      Y.Matrikelnummer
```

- Wie die Anfrage formulieren bei mehr als zwei Studenten?





order-by Klausel (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- Menge von Tupeln → Liste.
- Syntax
 - `order by attributliste`
- Beispiel
 - `select Matrikelnummer, Note`
 - `from Prüft`
 - `where V_Bezeichnung = 'Datenbanken I'`
 - `order by Note asc`
- Aufsteigend (**asc**)
oder absteigend (**desc**) sortieren.





order-by Klausel (2)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Sortierung wird auf das Ergebnis der jeweils vorangehenden SFW-Anfrage angewendet, funktioniert (in den meisten DBMS) aber auch auf Attribute, die nicht im select vorkommen:

```
select Matrikelnummer
```

```
from Prüft
```

```
where V_Bezeichnung = 'Datenbanken I'
```

```
order by Note
```

- Note kommt nicht in select-Klausel vor.





Behandlung von Nullwerten (1)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- 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**.





- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- Quantoren/Selbstverbund
- order-by
- [Nullwerte](#)
- Änderungen
- Schluß

```
aterm
mysql> select 'A'='A';
+-----+
| 'A'='A' |
+-----+
|         1 |
+-----+
1 row in set (0.01 sec)

mysql> select 'A'='B';
+-----+
| 'A'='B' |
+-----+
|         0 |
+-----+
1 row in set (0.00 sec)

mysql> select 'A'=NULL;
+-----+
| 'A'=NULL |
+-----+
|         NULL |
+-----+
1 row in set (0.00 sec)

mysql> select NULL=NULL;
+-----+
| NULL=NULL |
+-----+
|         NULL |
+-----+
1 row in set (0.00 sec)
```



Behandlung von Nullwerten (2)

Einleitung

SQL-Kern

Weitere
Konstrukte

- Übersicht

- Operationen

- Aggregation

- Gruppierung

- Quantoren/
Selbstverbund

- order-by

- Nullwerte

Änderungen

Schluß

- Beispiel: Stadt

Name	Bundesland
Frankfurt	Hessen
München	Bayern
Zürich	NULL

- **select** Name **from** Stadt
where Bundesland == 'Hessen'
- **select** Name **from** Stadt
where not (Bundesland == 'Hessen')



Behandlung von Nullwerten (3)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
 - Übersicht
 - Operationen
 - Aggregation
 - Gruppierung
 - Quantoren/Selbstverbund
 - order-by
 - Nullwerte
- Änderungen
- Schluß

- Selbst ein Vergleich $A=A$ ist bei Vorliegen von Nullwerten keine Tautologie mehr, sondern ergibt unknown. Derartiger Vergleich in where-Klausel also nicht eliminierbar.
- **select * from Stadt**
where Bundesland=Bundesland
nicht dasselbe wie
select * from Stadt

Stadt

Name	Bundesland
Frankfurt	Hessen
München	Bayern
Zürich	NULL

- Boolesche Ausdrücke → dreiwertige Logik



Behandlung von Nullwerten (4)

<i>and</i>	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

<i>or</i>	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

<i>not</i>	
true	false
unknown	unknown
false	true

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Übersicht
- Operationen
- Aggregation
- Gruppierung
- Quantoren/Selbstverbund
- order-by
- Nullwerte
- Änderungen
- Schluß





Änderungsoperationen

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- Einfügen von Tupeln in Basisrelationen (oder Sichten): **insert**,
- Löschen von Tupeln aus Basisrelationen (oder Sichten): **delete**,
- Ändern von Tupeln in Basisrelationen (oder Sichten): **update**,
- Diese Operationen jeweils als
 - Eintupel-Operationen (etwa die Erfassung einer neuen Ausleiherung),
 - Mehrtupel-Operationen („Erhöhe das Gehalt aller Mitarbeiter um 4.5%.“)
- Ändern von Sichten
 - grundsätzlich – in gleicher Weise möglich.

update (1)

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- **Syntax:**

```
update basisrelation  
set attribut_1 = ausdr_1, ...,  
      attribut_n = ausdr_n  
[ where bedingung ]
```

- **Beispiele:** Mitarbeiter

Name	Gehalt
Böhm	60
Buchmann	30
Lockemann	90

- **update** Mitarbeiter **set** Gehalt=Gehalt+10
where Gehalt < 50



update (2)

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

Mitarbeiter

Name	Gehalt
Böhm	60
Buchmann	30
Lockemann	90

- **update** Mitarbeiter **set** Gehalt = 60
where Name = 'Buchmann'
- **Was leistet das folgende Statement?**
update Angestellte **set** Gehalt = 60





delete

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- Syntax:

```
delete from basisrelation  
[ where bedingung ]
```

- Beispiel:

```
delete from Ausleihe where Invnr = 4711
```

- Standardfall ist Löschen mehrerer Tupel:

```
delete from Ausleihe where Name =  
'Meyer'
```

- Löschen der gesamten Relation:

```
delete from Ausleihe
```

Nicht dasselbe wie

```
drop table Ausleihe
```





insert (1)

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- **insert into** basisrelation
[(attribut_1, ..., attribut_n)]
values (konstante_1, ..., konstante_n)
- **insert into** Buch (Invnr, Titel)
values (4867, 'Wissensbanken')
- **insert into** Buch
values (4867, 'Wissensbanken', '3-
87', 'Karajan')





insert (2)

- Einleitung
- SQL-Kern
- Weitere Konstrukte
- Änderungen
- Schluß

- **insert into** basisrelation
[(attribut_1, ..., attribut_n)]
SQL-anfrage
- **insert into** Kunde (**select** LName, LAdr, 0
from Lieferant)

Kunde

Name	Adr	AnzahlAufträge
...

Lieferant

LName	LAdr	Produkt
...





SQL-Versionen

Einleitung

SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- Geschichte:
 - SEQUEL (1974, IBM Research Labs San Jose),
 - SEQUEL2 (1976, IBM Research Labs San Jose),
 - SQL (1982, IBM),
 - ANSI-SQL (SQL-86; 1986),
 - ISO-SQL (SQL-89; 1989; drei Sprachen Level 1, Level 2 und IEF),
 - (ANSI/ISO) SQL2, als SQL-92 verabschiedet,
 - (ANSI/ISO) SQL3, als SQL:99 verabschiedet,
 - (ANSI/ISO) SQL4 (geplant).





SQL-89

Einleitung


SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- Level 1:
 - Keine Nullwerte,
 - keine Selektionsbedingungen mit \neq oder **exists**,
 - keine **union**-Operation.
- Level 2: Wie hier beschrieben.
- Level 2 + IEF (Integrity Enhancement Feature):
 - **check**-Klausel: **where**-Klausel als Integritätsbedingung,
 - Definition von Primärschlüsseln und Fremdschlüsseln.



SQL-92 (1)

Einleitung


SQL-Kern

Weitere
Konstrukte

Änderungen

Schluß

- Neue Datentypen (wie `interval`),
- Domänenkonzept (**create domain**, **alter domain**),
- Änderung des Datenbankschemas: **alter table** und **drop table**,
- allgemeine Integritätsbedingungen (mehrere Tabellen),
- `string`-Operationen erweitert,
- Namen für abgeleitete Spalten,
- **join**, **cross join**, **natural join**, **outer join** als eigene Operatoren,
- auch **intersect** und **except**.



SQL-92 (2)

[Einleitung](#)

[SQL-Kern](#)

[Weitere
Konstrukte](#)

[Änderungen](#)

[Schluß](#)

- Sprache fast vollständig orthogonal, etwa **union**, SFW hinter **from**,
- dreiwertige Logik,
- **set transaction**: Verschiedene Isolationsstufen,
- Embedded SQL und Dynamic SQL sind Teil der Norm (siehe Abschnitt „Anwendungsprogrammierung“),
- Data Dictionary ist Teil der Norm.