



Kommunikation und Datenhaltung

Relationale Datenbanksprachen
(SQL – Structured Query Language)



Überblick über den Datenhaltungsteil

- Einleitung
 - Motivation und Grundlagen
 - Architektur von Datenbanksystemen
- Datenbankabfragen
 - Relationenmodell und Relationenalgebra
 - Relationale Datenbanksprachen (SQL)
- Datenbankentwurf
 - ER- und EER-Modell
 - Abbildung von ER-Modellen auf das Relationenmodell
 - Relationaler Entwurf
 - Sprachen zur Datenbankdefinition
- Transaktionsverwaltung
- Anfrageoptimierung
- Datenbankanwendungsentwicklung



Deklarative Sprachen

- Schwäche der Relationalen Algebra:
 - Für den ungeschulten Benutzer zu unhandlich.
- Mengenorientierung und damit Relationales Modell bildet aber natürliche Grundlage für **deklarative Sprachen**:
 - Deklarative Sprachen beschreiben lediglich gewünschtes Ziel (benutzernah).
 - Imperative Sprachen beschreiben Weg zum Ziel (implementierungsnah).
- An der Nutzerschnittstelle daher Ersatz der imperativen relationalen Algebra durch deklarative Sprache SQL.

Einleitung
SQL-Kern
Erweiterungen
Änderungen



Nutzung Deklarativer Sprachen

- Ad-hoc-Anfragesprachen für interaktiven Dialog zwischen menschlichem Nutzer und DBMS.
- In herkömmliche Programmiersprache eingebettete DML für Weiterverarbeitung der Daten durch Anwendungsprogramme.
- Datenbankprogrammiersprache mit vollständiger Integration des Datenmodells einschließlich DDL in eine Programmiersprache.
- In jedem Fall Sprachkonstrukte für Transaktionsprozeduren erforderlich: Beginn und Abschluss der Ausführung, Abbruch mit Rückkehr zum Ausgangszustand.

Einleitung
SQL-Kern
Erweiterungen
Änderungen



Structured Query Language (SQL)

- Anfragesprache für relationale DBMS
 - Standardisiert von ANSI und ISO
 - Heute von allen relationalen DBMS unterstützt
- Mehrere Teile
 - Datendefinition
 - Sichtdefinition
 - Anfrageteil
 - Formulierung von Datenänderungen
 - Definitionsteil für Dateiorganisation und Zugriffspfade
 - Teil mit imperativen Konstrukten zur Definition von Prozeduren und Funktionen

Einleitung
SQL-Kern
Erweiterungen
Änderungen



Der Sprachstandard SQL

- SQL-86 (1986): Erster Standardisierungsversuch
- SQL-89 (1989): Zweiter Standardisierungsversuch, immer noch mit Lücken
- SQL-92 (1992): Verbreiteter Standard
 - Sehr umfangreiches Werk (Normungsdokument > 600 Seiten ohne Beispiele)
- SQL-99 (1999, SQL-99): Nur teilweise in Produkten
 - Erweiterungen vor allem in Richtung Objektorientierung („objektrelationales“ Modell)
 - Wiederum: umfangreiches Werk (Normungsdokument > 1000 Seiten ohne Beispiele)
- SQL:2003
 - Für eine detailliertere geschichtliche Betrachtung siehe beispielsweise Saake, Sattler, Heuer, Kap. 10.6
- Wichtige Botschaft:
 - Es gibt nicht ein „SQL“
 - Trotz Standardisierung: Standards in unterschiedlichem Umfang umgesetzt; teilweise Inkompatibilitäten zwischen Systemen der einzelnen Hersteller; proprietäre Lösungen

Einleitung
SQL-Kern
Erweiterungen
Änderungen



Organisation dieses Kapitels/ Abgrenzung

- Betrachtung des Anfrageteils sowie der Konstrukte für Änderungen
- Zunächst: Kern, der Relationenalgebra abdeckt
 - SFW-Block
 - Mengenoperationen
 - Wie bei Algebra: Konstrukte sind „orthogonal“ – können kombiniert werden („Rechnen auf Tabellen“)
- Dann: Erweiterungen
 - Gruppierungen, Aggregation, usw.
 - Aber auch neuere Konstrukte (Rekursion)

Einleitung
SQL-Kern
Erweiterungen
Änderungen

Grundstruktur einer SQL-Anfrage

- Grundstruktur eines SQL-Ausdrucks:

```
select  A1, A2, ..., An
from    R1, R2, ..., Rm
[where  B]
```

} „SFW-Block“

- Gedankliches Abarbeitungsmodell:

- Bilde das kartesische Produkt der Relationen in der **from**-Klausel,
- wähle hieraus die Tupel aus, die die Bedingung B erfüllen,
- und projiziere das so erhaltene Ergebnis auf die in der **select**-Klausel aufgeführten Attribute.

- Äquivalenter Ausdruck der relationalen Algebra:

$$\pi_{A_1, A_2, \dots, A_n} (\sigma_B (R_1 \times R_2 \times \dots \times R_m)) .$$

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Laufendes Beispiel

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 |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

from-Klausel

- Syntax:
`select *`
`from relationenliste`
- Beispiel:
`select *`
`from BÜCHER`
- entspricht (trivialem) relationenalgebraischem Ausdruck:
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 |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Kartesisches Produkt

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

```
select *  
from AUSLEIH, BÜCHER
```

| Invnr | Name | Invnr | ... | Autor |
|-------|--------|-------|-----|---------|
| 4711 | Meyer | 0007 | ... | Fleming |
| 4711 | Meyer | 1201 | ... | Heuer |
| 4711 | Meyer | 4711 | ... | Vossen |
| ... | ... | ... | ... | ... |
| 1201 | Schulz | 0007 | ... | Fleming |
| 1201 | Schulz | 1201 | ... | Heuer |
| ... | ... | ... | ... | ... |
| 4711 | Meyer | 4717 | ... | Wirth |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Kartesisches Produkt

- Neben der impliziten Form kann man das Kreuzprodukt auch explizit formulieren:

```
select *  
from AUSLEIH cross join BÜCHER
```

| Invnr | Name | Invnr | ... | Autor |
|-------|--------|-------|-----|---------|
| 4711 | Meyer | 0007 | ... | Fleming |
| 4711 | Meyer | 1201 | ... | Heuer |
| 4711 | Meyer | 4711 | ... | Vossen |
| ... | ... | ... | ... | ... |
| 1201 | Schulz | 0007 | ... | Fleming |
| 1201 | Schulz | 1201 | ... | Heuer |
| ... | ... | ... | ... | ... |
| 4711 | Meyer | 4717 | ... | Wirth |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Die select-Klausel

- Abschließende Projektion

- Syntax:

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

- Bestandteile:

- Attribute aus from-Relationen,
- arithmetische Ausdrücke über Attribute und Konstanten,
- Aggregatfunktionen über Attributen.

- distinct: Ergebnismenge statt Multimenge
→ Duplikateliminierung

- SQL hat Multimengen-Semantik (als Voreinstellung) – warum?

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Projektion: Menge oder Multimenge

```
select Name  
from AUSLEIH
```

NAME

Meyer
Schulz
Müller
Meyer

```
select distinct Name  
from AUSLEIH
```

NAME

Meyer
Schulz
Müller

- entspricht

$\pi_{\text{Name}}(\text{AUSLEIH})$

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Die where-Klausel

- Selektion
- Syntax:

```
select ...from ...where bedingung
```

- Bedingung:

- *Konstanten-Selektion:*

```
attribut  $\theta$  konstante
```

- *Attribut-Selektion* zwischen Attributen
mit kompatiblen Wertebereichen:

```
attribut1  $\theta$  attribut2
```

- Mögliche Operatoren θ abhängig vom Wertebereich

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Konstantenselektion

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

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Ausleih

| INV.NR | NAME |
|--------|-------|
| 4711 | Meyer |
| 4712 | Meyer |

- entspricht

$$\pi_{\text{INV_NR, NAME}} (\sigma_{\text{NAME} = \text{'Meyer'}} (\text{AUSLEIH}))$$



Bereichsselektion

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

- Abkürzung für

```
attribut ≥ konstante1 and  
attribut ≤ konstante2
```

- Beispiel:

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

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Ungewißheitsselektion I

- Theoretisch nur Abkürzung für disjunktiv verknüpfte Bedingung.

- Syntax:

attribut **like** spezialkonstante

- Spezialkonstante kann beinhalten
 - '%' – kein oder beliebig viele Zeichen,
 - '_' – genau ein Zeichen.
 - SQL:2003 definiert “similar to”
 - Reguläre Ausdrücke in der Spezialkonstanten

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Ungewißheitsselektion II

- 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 ...
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Weitere Bedingungen

- *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).

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Verbund (Natürlicher Verbund) I

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 |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

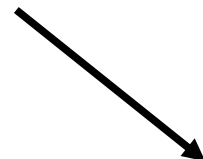
Verbund (Natürlicher Verbund) II

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 |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen



Natürlicher Verbund

- Frühere SQL-Versionen
 - Kennen nur Kreuzprodukt, keinen expliziten Verbundoperator (Verbund = Kreuzprod. & Sel.)
 - Verbund durch Prädikat hinter **where** realisieren
 - Zugleich mächtige Formulierung (erlaubt die Formulierung des θ -Verbunds und damit aller denkbaren Verbunde – s.u.)

- Beispiel:

```
select Name, Titel  
from AUSLEIH, BÜCHER  
where BÜCHER.INV_NR = AUSLEIH.INV_NR
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen



Verbund explizit: natural join

- Neuere SQL-Versionen
 - Kennen mehrere explizite Verbund-Operatoren
 - Als Abkürzung für die ausführliche Frage mit Kreuzprodukt aufzufassen
 - Übersichtlicher
 - Weniger fehleranfällig (Vergessen von Attributen)
 - Semantik entspricht dem Algebraoperator (insbes. werden doppelte Attribute eliminiert)

- Beispiel:

```
select Name, Titel  
from AUSLEIH natural join BÜCHER
```

- Äquivalent zu:

```
select Name, Titel  
from AUSLEIH join BÜCHER  
using (INV_NR)
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Theta Join (θ -Verbund) – Beispiel

Prof

| PName | PVorname |
|---------------|----------|
| Saake | Gunter |
| Rautenstrauch | Claus |
| Böhm | Klemens |

DKE

| Vorname | Name |
|---------|-----------|
| Klemens | Böhm |
| Erik | Buchmann |
| Ralf | Duckstein |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen



| 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 \geq , \neq , \approx usw. möglich.



Theta Join (θ -Verbund)

```
select *  
from Prof, DKE  
where PVorname < Vorname
```

```
select *  
from Prof join DKE  
    on (Prof.PVorname < DKE.Vorname)
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Self-Join I

- Self-Join: `select * from Person, Person`
 - Kartesisches Produkt einer Relation mit sich selbst, z.B. für tupelübergreifende Selektion

- 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 |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Self-Join II

- Einführung von Tupelvariablen:
Etwa auf eine Relation mehrfach zugreifen.

```
select * from Person eins, Person zwei
```

- Ergebnis hat vier Spalten:

```
eins.Name, eins.Vorname,  
zwei.Name, zwei.Vorname
```

- Was bestimmt diese Beispielanfrage über eine Relation `Person{Name,Alter}`?

```
select * from Person eins, Person zwei  
where eins.Alter < zwei.Alter  
      and eins.Name = 'Buchmann'
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Tupelvariablen und Relationennamen für Eindeutigkeit I

- Beispiel (Schemata):
WEINE(WeinID, Name, Farbe, Jahrgang, Weingut)
ERZEUGER(Name, Anbauggebiet, Region)

- Weingut ist Name des Erzeugers
(Fremdschlüssel in Relation WEINE)

aus welcher
Relation stammt
Name?

```
select Name, Jahrgang, Anbauggebiet falsch  
from WEINE, ERZEUGER  
where WEINE.Weingut = ERZEUGER.Name
```

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen



Tupelvariablen und Relationennamen II

- Attributnamen durch Präfix ergänzen:

```
select ISBN from Bücher
```

und

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

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Tupelvariablen und Relationennamen III

- Beispiel (Schemata):
WEINE(WeinID, Name, Farbe, Jahrgang, Weingut)
ERZEUGER(Name, Anbaugebiet, Region)
- Weingut ist Name des Erzeugers
(Fremdschlüssel in Relation WEINE)

richtig

```
select WEINE.Name, Jahrgang, Anbaugebiet  
from WEINE, ERZEUGER  
where WEINE.Weingut = ERZEUGER.Name
```

Wie sieht eine äquivalente Lösung mit Tupelvariablen aus?

Mengenoperationen

- Mengenoperationen erfordern kompatible Wertebereiche für Paare korrespondierender Attribute:
 - beide Wertebereiche sind gleich oder
 - beide sind auf **character** basierende Wertebereiche (unabhängig von der Länge der Strings) oder
 - beide sind numerische Wertebereiche (unabhängig von dem genauen Typ) wie **integer** oder **float**

- Ergebnisschema := Schema der „linken“ Relation

SFW_block1 **op** SFW_block2

- 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.

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Vereinigung, Durchschnitt und Differenz

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

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

- Äquivalent zu:

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

- **corresponding-Klausel:** Zwei Relationen nur über ihre gemeinsamen Attribute vereinigen, d. h. alle anderen Bestandteile fallen weg.

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Mengenoperationen II

- Über **corresponding by**-Klausel: Angabe der Attributliste, über die Mengenoperation ausgeführt wird

```
select *  
from  
    Professoren  
union corresponding by (PANr) Studenten)
```

- Bei Vereinigung: Defaultfall ist Duplikateliminierung (**union distinct**); **ohne** Duplikateliminierung durch **union all**

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Varianten der Union-Operation

- Einleitung
- SQL-Kern
- from
- select
- where
- Verbund
- Mengenop
- Schachtelung
- Erweiterungen
- Änderungen

R

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 4 |

S

| A | C | D |
|---|---|---|
| 2 | 3 | 4 |
| 2 | 4 | 5 |

R union S

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 4 |
| 2 | 4 | 5 |

R union all S

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 4 |
| 2 | 3 | 4 |
| 2 | 4 | 5 |

R union corresponding S

| A | C |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 2 | 3 |

R union corresponding by (A) S

| A |
|---|
| 1 |
| 2 |



Orthogonalität

- Jeder SFW-Block kann auch hinter **from** Klausel eingesetzt werden (Orthogonalität).
- Beispiel:

```
select Ergebnis.NAME from (  
    select * from BÜCHER join AUSLEIH  
)  
as Ergebnis
```
- Tupelvariablen für Zwischenergebnisse
 - „Zwischenrelationen“ aus SQL-Operationen oder einem SFW-Block können über Tupelvariablen mit Namen versehen werden
 - **as** ist optional

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Schachtelung von Anfragen

- Weitere Möglichkeit zur „Schachtelung“ (d.h. Einsetzen von SFV-Blöcken in Anfragen):
 - **where**-Klausel kann geschachtelt werden,
 - Im Folgenden betrachtet
 - Im aktuellen Standard auch select-Klausel
 - Problem bei Schachtelung in where-Klausel: SFV-Blöcke liefern im allgemeinen mehrere Werte
 - Standardvergleiche nicht anwendbar
 - Vergleiche mit Wertemengen:
 - Standardvergleiche in Verbindung mit Quantoren **all** (\forall) oder **any** (\exists),
 - spezielle Prädikate für den Zugriff auf Mengen **in** und **exists**.

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Das in-Prädikat I

- **Syntax:** `attribut in (SFW-block)`
- ‚true‘, falls aktuelles Tupel in Menge vorkommt
 - Attribut muss dabei kompatibel zum Ergebnisschema der inneren Anfrage sein

- **Beispiel:**

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

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 |

Empfiehlt

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

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Das in-Prädikat II

- Abarbeitung (konzeptionell):
 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')
```

- In der Praxis kann die Anfrage in eine Verbundanfrage umgeschrieben werden um sie effizient zu berechnen
- Wie sieht eine äquivalente Verbundanfrage für das Beispiel aus? (Keine Schachtelung!)

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

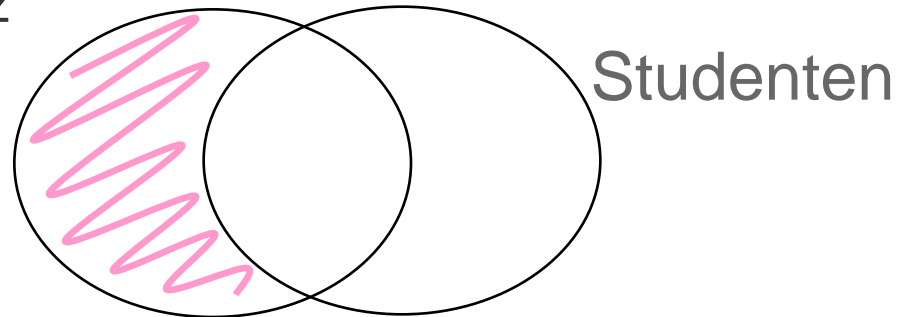
Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Exkurs: Simulation der Differenz

- In SQL-89 gab es als Mengenoperator nur die Vereinigung (keine Differenz oder Durchschnitt)

⇒ Andere Mengenoperationen mussten (können!) simuliert werden

- Beispiel: Differenz
Mitarbeiter



- In SQL:

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

- Gibt es hierzu äquivalente, nicht geschachtelte Anfrage?
 - Antwort erfordert Kenntnis weiterer (im Folgenden besprochener) Konstrukte

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Verzahnt geschachtelte Anfragen

- Bisher: Innere Anfrage konnte unabhängig von der äußeren betrachtet werden (→ konzeptionelle Abarbeitung)
- *Verzahnt geschachtelt*: Innere Anfrage hat Bezug zur Äußeren (referenziert Äußere via Relationennamen oder Tupelvariablen)
- Beispiel:

| Personen | | Prüft | | | |
|----------|-----|-------|-----|-----|-----|
| | | | | | |
| ... | ... | ... | ... | ... | ... |

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

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Verzahnt geschachtelte Anfragen: 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).

3. Ergebnis der inneren Anfrage in die äußere einsetzen. `1.0 in (2.0, 2.3)` ergibt **false**. Ersten Prüfer nicht berücksichtigen.
4. In der äußeren Anfrage das zweite `Personen`-Tupel untersuchen usw.

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

Verzahnt geschachtelte Anfragen II

Personen

| PANr | Name |
|------|------|
| ... | ... |

Prüft

| PANr | Matrikel | Vorlesung | Note |
|------|----------|-----------|------|
| ... | ... | ... | ... |

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

- Wie sieht eine äquivalente Anfrage ohne Schachtelung aus?

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

Das exists-Prädikat I

- 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

| ... | ... | ... |
|-----|-----|-----|

Ausleihe

| ... | ... | ... | ... |
|-----|-----|-----|-----|

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

Das exists-Prädikat II

- Formulieren Sie die Anfrage

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

- Hintergrund: Simulation des Allquantors durch exists

$$\forall \phi \equiv \neg \exists \neg \phi$$

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

exists: Simulation des Allquantors I

- 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.“

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

exists: Simulation des Allquantors II

- Beispiel:

```
select Lehrstuhl  
from Professoren  
where not exists
```

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

```
( 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) )
```

Mächtigkeit des SQL-Kerns

Jede Operation der Relationenalgebra durch eine oder mehrere Klauseln des SQL-Anfrageteils abgedeckt

Einleitung
SQL-Kern
from
select
where
Verbund
Mengenop
Schachtelung
Erweiterungen
Änderungen

| Relationenalgebra | SQL |
|-------------------|--|
| Projektion | select distinct |
| Selektion | where ohne Schachtelung |
| Verbund | from, where from mit join oder natural join |
| Umbenennung | Tupelvariable aus from -Klausel as für Attribute in select -Klausel (kommt noch!) |
| Differenz | where mit Schachtelung except corresponding |
| Durchschnitt | where mit Schachtelung intersect corresponding |
| Vereinigung | union corresponding |



Weitere Sprachkonstrukte von SQL

- Anfrageteil von SQL bietet viele über die Relationenalgebra hinausgehende Möglichkeiten
- Erweiterungen des SFW-Blocks
 - Aggregatfunktionen (in select-Klausel)
 - Zusätzliche Klauseln group by und having
 - Innerhalb der from-Klausel weitere Verbundoperationen (äußerer Verbund),
 - Innerhalb der where-Klausel weitere Arten von Bedingungen und Bedingungen mit Quantoren,
 - Innerhalb der select-Klausel die Anwendung von skalaren Operationen
- Rekursive Anfragen

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Aggregatfunktionen

- Beispiele:

```
select count (Invnr)  
from AUSLEIH
```

```
select  
count (distinct Name)  
from AUSLEIH
```

- Ergebnisse:

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

Ausleih

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Aggregatfunktionen I

- Aggregatfunktionen in Standard SQL:
 - **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
- SQL-Erweiterungen beinhalten zusätzliche Aggregatfunktionen (Statistik, Physik).

Nur bei numerischen Wertebereichen

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Aggregatfunktionen II

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Aggregatfunktionen: Beispiele

- **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)
 - Da Aggregatfunktionen nur einzelne Werte zurückliefern kann man sie in Konstantenselektionen der where-Klausel einsetzen

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Aggregate über Gruppierungen I

- Bisher: Aggregatfunktionen liefern einen Wert pro Relation
- Verfeinerung: Aggregate für Teile der Relation (Gruppen)

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Aggregate über Gruppierungen II

| 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(Anzahl)

| Marke | Anzahl |
|-------|--------|
| BMW | 106 |
| Opel | 210 |
| Audi | 196 |

Marke
→
avg(Anzahl)

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

Marke
→
max(Anzahl)

| Marke | Anzahl |
|-------|--------|
| BMW | 41 |
| Opel | 95 |
| Audi | 62 |

Bundesland
→
sum(Anzahl)

| Bundesland | Anzahl |
|------------|--------|
| Hessen | 131 |
| Bayern | 240 |
| Saarland | 141 |

Aggregate über Gruppierungen III

| 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(Anzahl)

| Marke | Bundeslar | 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 - Beispiele I

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

group by - Beispiele II

- Beispiele von eben (Forts.):

```
select Marke, Bundesland, sum(Anzahl) as Anzahl  
from Zulassungen  
group by Marke, Bundesland
```

- Ist diese Anfrage **auch** zulässig?

```
select Marke, Datum, sum(Anzahl) as Anzahl  
from Zulassungen  
group by Marke, Bundesland
```

- Antwort erfordert Wissen folgender Folien

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

group by und having

- Syntax

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

- Beispiel:

```
select Bundesland, count(*)  
from Zulassungen  
group by Bundesland  
having sum(Anzahl) > 100
```

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Abarbeitungsmodell bei Gruppierung

- Werte **from**- und **where**-Klauseln aus wie bisher (Bildung von Kart. Produkt und Selektion).
- Fasse in der Ergebnisrelation alle Tupel, die gleiche Werte für die Gruppierungsattribute aufweisen, zu einer Gruppe zusammen.
- Werte nun für jede einzelne Gruppe die **select**-Klausel aus. Dadurch entsteht für jede Gruppe genau ein Tupel.
- Falls eine **having**-Klausel angegeben wurde, werte diese für jede Gruppe aus und eliminiere nicht-qualifizierende Gruppen.
- Wenn **select distinct** angegeben wurde, eliminiere Duplikate aus der Menge der so entstandenen Tupel.

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

group by und having

- Syntax

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

- **having** ist Selektionsbedingung auf gruppierter Relation, darf Bezug nehmen auf:

- Gruppierungsattribute,
- beliebige Aggregatfunktionen über Nicht-Gruppierungsattributen.
- Hintergrund: Attribute in der having-Klausel müssen pro Gruppe eindeutig sein
(da having pro Gruppe selektiert, d.h., es wird pro Gruppe ein Tupel erzeugt – anschließend wird für jede Gruppe entschieden, ob ihr Tupel Teil des Ergebnisses ist)

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Veranschaulichung des Abarbeitungsmodells

- 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 |

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

- Anfrage:

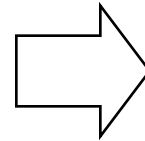
```
select A, sum(D) from REL where B<4  
group by A, B  
having A<4 and sum(D)<10 and max(C)=4
```

Gruppierung: Schritt 1

- from REL where B<4

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

| 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 |

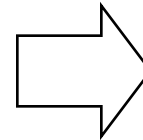
```
select A, sum(D) from REL where B<4  
group by A, B  
having A<4 and sum(D)<10 and max(C)=4
```

Gruppierung: Schritt 2

- **group by A, B**

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

| 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 |

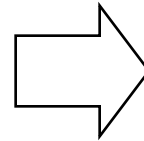
```
select A, sum(D) from REL where B<4
group by A, B
having A<4 and sum(D)<10 and max(C)=4
```

Gruppierung: Schritt 3

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

| 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 |

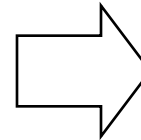
```
select A, sum(D) from REL where B<4
group by A, B
having A<4 and sum(D)<10 and max(C)=4
```

Gruppierung: Schritt 4

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

| 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 |

```
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: Beispiele I

- Basisrelation:

Ausleihe

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

Umbenennung eines Attributs – ohne diese Angabe wäre die entsprechende Spalte des Ergebnisses nicht benannt

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

| Anzahl | PANr |
|--------|------|
| 2 | 7754 |
| 1 | 4711 |
| 1 | 5588 |
| 2 | 9912 |

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Gruppierung: Beispiele II

- `select count(*) as Anzahl, 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)`

- Zugrunde liegende
Relation:

| Prüft | |
|----------------|------|
| Matrikelnummer | Note |
| ... | ... |

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

SQL/99: Allgemeine select-Syntax (1)

- Allgemeine Form der **select**-Anweisung:

```
select [distinct]   item1 [[as] A1], ..., itemk [[as] Ak]  
from                table-expr1 [[as] R1], ..., table-expri [[as] Ri]  
[where              cond-expr  
[group by          col1, ..., colm]  
[having            cond-expr
```

- Dabei hat jedes *item*_{*i*} eine der Formen
 - *e*_{*i*}, wobei *e*_{*i*} ein Attributname oder ein arithmetischer Ausdruck ist,
 - *R*_{*i*}.* (Auswahl aller Attribute von *R*_{*i*}),
 - *f*(*e*_{*i*}) mit *f* ∈ {**count**,**sum**,**avg**,**min**,**max**} und *e*_{*i*} wie oben,
 - *f*(**distinct** *e*) mit *f* und *e* wie oben,
 - **count**(*).
- „*“ statt *item*-Liste: Auswahl aller Spalten des durch **from** definierten Kreuzprodukts.

SQL/99: Allgemeine select-Syntax (2)

- Allgemeine Form der **select**-Anweisung:

```
select [distinct]   item1 [[as] A1], ..., itemk [[as] Ak]  
from                table-expr1 [[as] R1], ..., table-expri [[as] Ri]  
[where              cond-expr  
[group by          col1, ..., colm]  
[having            cond-expr
```

- Einschränkungen und Default-Regeln:

- Weglassen der **where**-Klausel ist äquivalent zu „true“.
- Bei Angabe einer **group by**-Klausel dürfen in der **select**- und **having**-Klausel Attribute, die von *col*₁, ..., *col*_{*m*} verschieden sind, nur als Argumente von Aggregatfunktionen auftreten.
- Bei Angabe von Aggregatfunktionen ohne **group by**-Klausel wird implizit eine **group by**-Klausel mit leerer Attributliste angenommen, d.h., die ganze Ergebnistabelle wird als eine Gruppe betrachtet.



Allgemeine Form der Sortierklausel

- Ergebnis eines Tabellenausdrucks kann für interaktive Präsentation oder programmgesteuertes Auslesen sortiert werden, indem hinter Ausdruck eine Klausel

order by col_1 [**asc** | **desc**], ..., col_m [**asc** | **desc**]

mit col_i Attributname oder Spaltennummer eingefügt wird.

- Schlüsselwort **asc** oder **desc** gibt aufsteigende (Default) bzw. absteigende Sortierfolge an.
- Beachte: Sortierung ist *Eigenschaft der Präsentation*, nicht der gespeicherten Tabellen.
 - Relation ist UNGEORDNETE Menge

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

order-by Klausel I

- Syntax

```
order by attributliste
```

- Beispiel 1:

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

- Beispiel 2 (berechnetes Attr. als Sortierkriterium):

```
select Note, count(*) as Anzahl  
from Prüft  
where V_Bezeichnung = 'Datenbanken I'  
group by Note  
order by Anzahl desc
```

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

order-by Klausel II

- Sortierung wird auf das Ergebnis der jeweils vorangehenden SFW-Anfrage angewendet

⇒ Sortierattribut muss in der select-Klausel vorkommen!

- Funktioniert in den meisten DBMS aber auch auf Attribute, die nicht in select-Klausel vorkommen:

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

- Note kommt nicht in select-Klausel vor

⇒ Lt. Standard falsch!

Sortierung: Top-k Anfragen

- Anfrage, die die besten k Elemente bzgl. einer Rangfunktion liefert („Abschneiden“ der Ergebnisliste)
- Vorgehen mit Hilfe eines Self-Joins:
 - Schritt 1: Zuordnung aller Tupel die größer oder gleich sind (bei aufsteigender Sortierung)
 - Schritt 2: Berechnung des Rangs mittels Gruppierung und Aggregation
 - Schritt 3: Beschränkung auf k Ränge (having)
 - Schritt 4: Sortierung nach Rang

- Beispiel:

```
select s1.Name, s1.Matrikelnummer, count(*) as Rank
from Student s1, Student s2
where s1.Matrikelnummer >= s2.Matrikelnummer -- Schritt 1
group by s1.Name, s1.Matrikelnummer -- Schritt 2
having count(*) <= 4 -- Schritt 3
order by Rank -- Schritt 4
```

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Quantoren und Mengenvergleiche I

- Hintergrund (Wdh.): Schachtelung in der where-Klausel erfordert Vergleich mit Mengen (basierend auf Quantoren)
 - in und exists wurden bereits besprochen
 - Jetzt allgemeine Form

- **Syntax**

```
attribut  θ{ all | any | some }  
                ( select attribut  
                  from ... where ... )
```

für einen Standardvergleichsoperator θ

- Bedeutung: **all** Allquantor,
any, **some** Existenzquantoren (äquivalent)

- Beispiel:

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

„= any“ entspricht „in“

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Quantoren und Mengenvergleiche II

- **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?

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Quantoren und Mengenvergleiche III

- Anwendbarkeit eingeschränkt:
Test auf Mengen-Gleichheit geht nicht.
 - Definition von Mengen-Gleichheit:
$$\forall x \in M_1: \exists x \in M_2 \wedge \forall x \in M_2: \exists x \in M_1$$
 - Grund: äußere Anfrage wird tupelweise abgearbeitet
- Beispiel: In SQL so nicht umsetzbar:
Gib alle Bücher aus, an denen 'Heuer' und 'Saake' gemeinsam als Autoren beteiligt waren.
- Umsetzung erfordert einen Self-Join
 - Formulieren Sie die Anfrage

Bücher

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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Äußere Verbunde I

- Klassischer Verbund: **[inner] join**
 - Tupel ohne Verbundpartner (“dangling tuples”) gehen nicht ins Ergebnis ein
 - Manchmal aber interessant – Beispiel: Anzahl ausgeliehener Bücher pro Autor (dabei wollen wir also auch wissen, von welchen Autoren 0 Bücher verliehen sind)
- **outer join**
 - dangling tuples übernehmen und mit Nullwerten auffüllen
 - **left outer join**: Im linken Operanden
 - **right outer join**: Im rechten Operanden
 - **full outer join**: In beiden Operanden
 - Syntax wie beim **inner join** (inkl. **natural**, **using**, **on**)
- Beispiel:

```
select Autor, count(INV_NR) as Anzahl
from BÜCHER left outer join AUSLEIH using (INV_NR)
group by Autor
```

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Äußere Verbunde II

| 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 |

- Einleitung
- SQL-Kern
- Erweiterungen
- Aggregation
- Gruppierung
- Order By
- Weitere Ops/Prädikate
- Null Werte
- Rekursion
- Änderungen

Basisrelationen

```
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

```
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> █
```

Vereinigung und äußere Verbunde

- Äußerer Verbund ist eine abgeleitete Operation, d.h., kann auch anders ausgedrückt werden

```
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, cast(null as string)  
from Personen P  
where not exists ( select *  
                    from Pers_Telefon T  
                    where P.PANr = T.PANr )
```

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Tupelbildungen I

- Zusätzliche Möglichkeit für Vergleichen innerhalb von where-Klauseln: Nutzung von *Tupelbildungen (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')
```

- Bei Tupelbildungen in Vergleich muss Anzahl der Attribute übereinstimmen; außerdem müssen Attribute kompatibel sein
 - Beispiel: (Böhm, Klemens, 39114) nicht vergleichbar mit (Böhm, Klemens, Zürich)

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Tupelbildungen II

- Definition des Vergleichsergebnisses für Tupel
 - = alle Komponenten müssen gleich sein
 - \neq mind. eine Komponente muss ungleich sein
 - $(a_1, \dots, a_n) < (b_1, \dots, b_n)$ – wahr, wenn ein j existiert, für das $a_j < b_j$ und $a_i = b_i$ für alle $i < j$
 - Beispiel: (Böhm, Klemens, Magdeburg) < (Böhm, Klemens, Zürich)
 - Entspricht lexikographischer Ordnung!

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Operationen auf Wertebereichen I

- Innerhalb von **select** und **where**:
Statt Attributen auch *skalare Ausdrücke*.

- Beispiel:

```
select ISBN, Preis / 1.19 as Netto  
from Bücher
```

- Ausdrücke werden tupelweise ausgewertet.

⇒ Ergebnis:

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

Umbenennung eines Attributs – ohne diese Angabe wäre die zweite Spalte des Ergebnisses nicht benannt

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Operationen auf Wertebereichen II

- Innerhalb von **select** und **where**:
Statt Attributen auch *skalare Ausdrücke*.
- Skalare Ausdrücke:
Folgen von Operationen über Attributen und Konstanten (kompatibler Wertebereich)
- Operationen:
 - Auf numerischen Wertebereichen: etwa + , - , * , /,
 - Auf Strings: **char_length**, Konkatination ||, **substring** (Teilzeichenkette),
 - Auf Datumstypen, Zeitintervallen: **current_date**, **current_time**, + , - , *.
 - Darüber hinaus: case-Anweisungen, Typcasts, etc.

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Behandlung von Nullwerten I

- Nullwert: Extra-Symbol
 - Gehört zu keinem Wertebereich
 - “Wert unbekannt”

- 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')
```

Ergebnisse?

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Behandlung von Nullwerten II

- Nullwert: Extra-Symbol
 - Gehört zu keinem Wertebereich
 - “Wert unbekannt”
- Wirkung von Nullwerten:
 - In skalaren Ausdrücken: Ergebnis **null**, sobald Nullwert in die Berechnung eingeht.
 - In allen Aggregatfunktionen bis auf **count(*)** werden Nullwerte vor Anwendung der Funktion entfernt.
 - Vergleiche mit Nullwert ergeben Wahrheitswert **unknown** statt **true** oder **false**.
 - U.a.: $(\text{null} = \text{null}) \equiv \text{unknown}$!
 - Ausnahme: **is null** ergibt **true**, **is not null** ergibt **false**

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Behandlung von Nullwerten III

- 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

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen



Behandlung von Nullwerten IV

| <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
- Erweiterungen
- Aggregation
- Gruppierung
- Order By
- Weitere Ops/
Prädikate
- Null Werte
- Rekursion
- Änderungen

Beispiel

```
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)
```

- Einleitung
- SQL-Kern
- Erweiterungen**
- Aggregation
- Gruppierung
- Order By
- Weitere Ops/
Prädikate
- Null Werte**
- Rekursion
- Änderungen

Rekursive Anfragen

Beispiel:

„Flüsse, die direkt oder indirekt in die Nordsee münden.“

Basisrelation: MündetIn(Fluss, FlussOderMeer)

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Rekursive Anfrage:

with recursive MündetIndirektIn (Fluss, FlussOderMeer) **as**

```
(select Fluss, FlussOderMeer
  from MündetIn
  where FlussOderMeer = 'Nordsee')
```

union

```
select MündetIn.Fluss, MündetIn.FlussOderMeer
  from MündetIn, MündetIndirektIn
  where MündetIn.FlussOderMeer = MündetIndirektIn.Fluss )
```

```
select Fluss
  from MündetIndirektIn;
```

Definition der rekursiv zu konstruierenden Relation (Sicht)

Basisschritt: Initialisierung der zu konstruierenden Relation

Rekursionschritt

Anfrage an die rekursive Sicht



Sicherheit rekursiver Anfragen

- Sicherheit (= Endlichkeit der Berechnung) ist wichtige Anforderung an Anfragesprache
- Problem: Zyklen bei Rekursion
 - Tritt in obigem Beispiel (wahrscheinlich) nicht auf (aufgrund der Anwendungssemantik)
 - **Beispiel, bei dem es zu Zyklen kommen kann?**
- Behandlung in SQL
 - Begrenzung der Rekursionstiefe
 - Zyklenerkennung
 - Beispiele: Saake, Sattler, Heuer, Kap. 10.6

Einleitung
SQL-Kern
Erweiterungen
Aggregation
Gruppierung
Order By
Weitere Ops/
Prädikate
Null Werte
Rekursion
Änderungen

Änderungsoperationen

- Einfügen von Tupeln: **insert**
- Löschen von Tupeln: **delete**
- Ändern von Tupeln: **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 Basisrelationen und Sichten in gleicher Weise
 - Allerdings bei Sichten nicht immer möglich

Einleitung
SQL-Kern
Erweiterungen
Änderungen

update I

- Syntax:

```
update relationenname  
set attribut_1 = ausdruck_1, ...,  
      attribut_n = ausdruck_n  
[ where bedingung ]
```

Einleitung
SQL-Kern
Erweiterungen
Änderungen

- Beispiel:

Mitarbeiter

| Name | Gehalt |
|-----------|--------|
| Böhm | 60 |
| Buchmann | 30 |
| Lockemann | 90 |

```
update Mitarbeiter set Gehalt = 60  
where Name = 'Buchmann,
```

- Attributänderungen sind jeweils Mehr-Tupel-Operationen
 - Simulation einer Ein-Tupel-Operation mittels where-Klausel

update II

Mitarbeiter

| Name | Gehalt |
|-----------|--------|
| Böhm | 60 |
| Buchmann | 30 |
| Lockemann | 90 |

Einleitung
SQL-Kern
Erweiterungen
Änderungen

- Alter Attributwert kann zur Berechnung des neuen Wertes herangezogen werden:

```
update Mitarbeiter set Gehalt=Gehalt+10  
where Gehalt < 50
```

- Was leistet das folgende Statement?

```
update Mitarbeiter set Gehalt = 60
```

delete

- Syntax:

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

Einleitung
SQL-Kern
Erweiterungen
Änderungen

- 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)

- Beachte: Löschen kann leicht zur Verletzung von Integritätsbedingungen führen und vom System zurückgewiesen werden (Beispiel?)

insert I

- Zwei unterschiedliche Formen des Einfügens:
 - Konstante Tupel
 - Berechnete Tupel (aus anderen Relationen)
- Syntax für konstante Tupel:

```
insert into basisrelation  
    [(attribut_1, ..., attribut_n)]  
values (konstante_1, ..., konstante_n)
```
- Optionale Attributliste für Einfügen unvollständiger Tupel
 - Nicht aufgeführte Tupel werden auf null gesetzt
 - Bei fehlender Liste müssen alle Attributewerte gesetzt werden
- Beispiele:
 - **insert into** Buch (Invnr, Titel)
values (4867, 'Wissensbanken')
 - **insert into** Buch
values (4867, 'Wissensbanken', '3-87', 'Karajan')

insert II

- Syntax für berechnete Tupel:

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

Einleitung
SQL-Kern
Erweiterungen
Änderungen

- Beispiel:

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

- Durch Einsetzen von Konstanten können berechnete Werte mit konstanten Werten kombiniert werden

Kunde

| Name | Adr | AnzahlAufträge |
|------|-----|----------------|
| ... | ... | ... |

Lieferant

| LName | LAdr | Produkt |
|-------|------|---------|
| ... | ... | ... |

Zusammenfassung

- Relationale Anfragesprache SQL
- SQL-Kern (deckt Relationenalgebra ab)
 - SFW-Block
 - Mengenoperationen
 - Konstrukte sind „orthogonal“ – können kombiniert werden („Rechnen auf Tabellen“)
- Erweiterungen
 - Aggregatfunktionen (in select-Klausel)
 - Zusätzliche Klauseln group by und having
 - Sortierung (order by)
 - Äußerer Verbund
 - where-Klausel: Bedingungen mit Quantoren,
 - select-Klausel: skalare Operationen
 - Nullwerte
 - Rekursion
- Änderungsoperationen