

# Kapitel DB: VI

- I. Einführung und grundlegende Konzepte von Datenbanken
- II. Datenbankentwurf und Datenbankmodelle
- III. Konzeptueller Datenbankentwurf
- IV. Logischer Datenbankentwurf mit dem relationalen Modell
- V. Grundlagen relationaler Anfragesprachen

## VI. Die relationale Datenbanksprache SQL

- ❑ Einführung
- ❑ SQL als Datenanfragesprache
- ❑ SQL als Datendefinitionssprache
- ❑ SQL als Datenmanipulationssprache
- ❑ Sichten
- ❑ SQL vom Programm aus

## VII. Entwurfstheorie relationaler Datenbanken

# Einführung

## Historie

- 1974 Ausgangspunkt ist SEQUEL, die *Structured English Query Language*
  - : Wildwuchs verschiedener Dialekte
- 1986 erste Festlegung eines Standards
- 1989 endgültiger Sprachstandard entsteht unter dem Namen SQL-89
- 1990 SQL-89 wird in Deutschland als DIN ISO 9075 publiziert
- 1992 nach Überarbeitung und Erweiterung wird **SQL-92 (SQL2)** verabschiedet
- 1993 SQL-92 wird in Deutschland als DIN 66315 publiziert
- 1999 SQL:1999 (SQL3) wird verabschiedet
- 2003 SQL:2003 wird von der ISO als Nachfolger von SQL:1999 verabschiedet
- 2006 SQL:2006 definiert die Verwendung von SQL in Zusammenhang mit XML
- 20xx SQL4

## Bemerkungen:

- ❑ SEQUEL wurde bei IBM-Research, San Jose, entwickelt. Es diente als Schnittstelle zum experimentellen relationalen Datenbanksystem „R“.
- ❑ SQL steht für *Structured Query Language*.

# Einführung

## Vergleich zu theoretischen Anfragesprachen

### Relationen in SQL:

- sind im Allgemeinen nicht duplikatfrei sondern Multimengen
- Duplikatfreiheit in Basisrelationen wird mit Integritätsbedingungen realisiert; in Ergebnisrelationen müssen Duplikate explizit entfernt werden.

### Anfragen in SQL:

- bilden die Relationenalgebra weitgehend ab
- besitzen Grenzen hinsichtlich der Orthogonalität
- enthalten zusätzlich Operationen zur Aggregation, Gruppierung, Sortierung, Verarbeitung spezieller Datentypen

### weitere Konzepte von SQL:

- Definition von Datenbanken
- Pflege und Modifikation von Relationen
- Verwaltung von Benutzern, Autorisierung

# Einführung

## Komponenten von SQL

1. Datendefinitionssprache, DDL.
2. Datenmanipulationssprache, DML.
3. Data Query Language, DQL.
4. Transaktionskontrolle.

# Einführung

## Komponenten von SQL

### 1. Datendefinitionssprache, DDL.

Definition und Modifikation der Datenstrukturen für Datenbanken:

- externe Ebene: Sichten und Zugriffsrechte (Autorisierung)
- konzeptuelle Ebene: Relationenschemata und Integritätsbedingungen
- physische Ebene: Indexe

### 2. Datenmanipulationssprache, DML.

Einfügen, Ändern und Löschen von Daten

### 3. Data Query Language, DQL.

Formulierung von Anfragen, Auswahl und Aufbereitung von Daten

### 4. Transaktionskontrolle.

Spezifikation von Transaktionen,  
Sperrungen von Daten für das Concurrency Control

# SQL als Datenanfragesprache

Kern von SQL-Anfragen ist der Select-From-Where-Block (SFW-Block) :

- `select`  
spezifiziert die Attribute des Ergebnisschemas
- `from`  
spezifiziert die verwendeten Relationen; das können Basisrelationen oder auch abgeleitete Relationen sein
- `where`  
spezifiziert Selektions- und Verbundbedingungen

# SQL als Datenanfragesprache

Kern von SQL-Anfragen ist der Select-From-Where-Block (SFW-Block) :

- `select`  
spezifiziert die Attribute des Ergebnisschemas
- `from`  
spezifiziert die verwendeten Relationen; das können Basisrelationen oder auch abgeleitete Relationen sein
- `where`  
spezifiziert Selektions- und Verbundbedingungen
  
- `group by`  
spezifiziert die Attribute, hinsichtlich derer Tupel gruppiert werden
- `having`  
spezifiziert Selektionsbedingung für Gruppen
- `order by`  
spezifiziert ein Prädikat zur Sortierung der Ergebnistupel
- `union`  
ermöglicht Vereinigung mit Ergebnistupeln nachfolgender SFW-Blöcke

# SQL als Datenanfragesprache

## Illustration der Grundideen

Buecher		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select *  
from Buecher
```

~>

ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

# SQL als Datenanfragesprache

## Illustration der Grundideen

Buecher		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Titel, Verlag  
from Buecher
```

~>

Titel	Verlag
Brecht Lesebuch	Piper
Längengrad	Berlin-Verlag
Fundamentals of Database Systems	Addison Wesley
Heuristics	Addison Wesley

# SQL als Datenanfragesprache

## Illustration der Grundideen

Buecher		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Titel, Verlag  
from Buecher  
where Verlag = 'Addison Wesley'
```

~>

Titel	Verlag
Fundamentals of Database Systems	Addison Wesley
Heuristics	Addison Wesley

# SQL als Datenanfragesprache

## Illustration der Grundideen

Buecher		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Verlag  
from Buecher
```

~>

Verlag
Piper
Berlin-Verlag
Addison Wesley
Addison Wesley

# SQL als Datenanfragesprache

## Illustration der Grundideen

Buecher		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Längengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select distinct Verlag  
from Buecher
```

~>

Verlag
Piper
Berlin-Verlag
Addison Wesley

# SQL als Datenanfragesprache

## Syntax des SFW-Blocks (vereinfacht)

```
select [all | distinct]  
  {*| <attribute1> [[as] <alias1>], <attribute2> [[as] <alias2>], ...}
```

```
from <table1> [[as] <alias1>], <table2> [[as] <alias2>], ...
```

```
[where <condition>]
```

```
[group by <attribute1>, <attribute2>, ...]
```

```
[having <condition>]
```

```
[order by <attribute1>, <attribute2>, ...[asc | desc]]
```

```
[union [all]]
```

```
[limit [<offset_num>,] <tuple_num>]
```

## Bemerkungen:

- ❑ `[[as] <alias>]` dient zur Deklaration zusätzlicher Bezeichner für Attribute und Tupelvariablen im lexikalischen Gültigkeitsbereich des SFW-Blocks.
- ❑ `<condition>` ist eine Formel, die aus Atomen und logischen Junktoren aufgebaut ist. Die Atome entsprechen weitgehend den Atomen im Tupel- und Domänenkalkül.
- ❑ Seit SQL-92 sind in der From-Klausel auch Join-Operatoren oder ein SFW-Block zugelassen, um eine neue (virtuelle) Relation aufzubauen.

# SQL als Datenanfragesprache

## From-Klausel

```
from <table1> [[as] <alias1>], <table2> [[as] <alias2>], ...
```

- Die From-Klausel spezifiziert die Relationen einer Anfrage und bildet somit den *Ausgangspunkt* für die Anfragebearbeitung.
- Eine Komma-separierte Liste von Relationen entspricht der Bildung des kartesischen Produktes.
- Die Verwendung von Aliasen entspricht der Einführung von *Tupelvariablen*, die zur *Qualifizierung* von Attributen verwandt werden können.
- Aliase ermöglichen u. a. die Spezifikation von denselben Attributen aus verschiedenen Tupeln derselben Relation zur Formulierung *tupelübergreifender Bedingungen*. Stichwort: Selbstverbund (Self-Join)

# SQL als Datenanfragesprache

## From-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

**select** \*

**from** Mitarbeiter eins, Mitarbeiter as employee

- ~> Bildung des kartesischen Produktes,  
Ausgabe einer Tabelle mit 10 Spalten und 16 Zeilen

# SQL als Datenanfragesprache

## Select-Klausel

```
select [all | distinct]  
  {*| <attribute1> [[as] <alias1>], <attribute2> [[as] <alias2>], ...}
```

- Die Select-Klausel spezifiziert die Attribute  $A_i$  des Ergebnisschemas. Die  $A_i$  müssen aus den in der From-Klausel spezifizierten Relationen  $r_j$  stammen. Mittels „\*“ (Wildcard) werden alle Attribute ausgewählt.
- Zur Unterscheidung gleichbenannter Attribute  $A$  in verschiedenen Relationen  $r_1, r_2$  ist eine *Qualifizierung mittels Tupelvariablen* möglich:  $r_1.A, r_2.A$ . Für jede Basisrelation  $r$  ist implizit eine Tupelvariable mit dem Namen der Relation vereinbart.
- Die Verwendung von Aliassen bedeutet eine Umbenennung von Attributen im Ergebnisschema.
- Das Schlüsselwort „distinct“ gibt an, ob die Tupel der Ergebnisrelation eine Menge oder eine Multimenge bilden.

## Bemerkungen:

- Die mittels „distinct“ erzwungene Duplikateliminierung ist nicht der Default. Gründe:
  - Duplikateliminierung erfordert in der Regel eine (aufwendige) Sortierung.
  - Bei Anfragen, die alle Tupel betreffen, kann die Eliminierung von Duplikaten zur Ergebnisverfälschung führen.

# SQL als Datenanfragesprache

## Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent:

```
select PersNr  
from Mitarbeiter
```

```
select Mitarbeiter.PersNr  
from Mitarbeiter
```

```
select M.PersNr  
from Mitarbeiter as M
```

# SQL als Datenanfragesprache

## Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent:

```
select PersNr                                select Mitarbeiter.PersNr
from Mitarbeiter                            from Mitarbeiter
```

```
select M.PersNr
from Mitarbeiter as M
```

Unerlaubte Anfrage wegen Mehrdeutigkeit von PersNr:

```
select PersNr
from Mitarbeiter, ArbeitetInProjekt
```

# SQL als Datenanfragesprache

## Where-Klausel

[ **where** <condition> ]

- Die Where-Klausel dient zur Selektion von Tupeln aus den Relationen, die in der From-Klausel spezifiziert sind. Alle Tupel, die <condition> erfüllen, werden in die Ergebnismenge aufgenommen.
- <condition> entspricht einer logischen Formel, vergleichbar der Formel  $\alpha$  im Tupel- oder Domänenkalkül [vgl. DB:V-69, DB:V-84]. Ausnahmen bzw. Ergänzungen sind u. a.:
  - die Junktoren heißen `and`, `or`, `not`
  - es gibt mengenwertige Operanden; die Operatoren hierfür sind `in`, `exists`, `any`
  - der Allquantor ist nicht zugelassen
  - ein Operand kann eine Unterabfrage, also wiederum ein komplexer SQL-Block sein

# SQL als Datenanfragesprache

## Where-Klausel

[ **where** `<condition>` ]

- Die Where-Klausel dient zur Selektion von Tupeln aus den Relationen, die in der From-Klausel spezifiziert sind. Alle Tupel, die `<condition>` erfüllen, werden in die Ergebnismenge aufgenommen.
- `<condition>` entspricht einer logischen Formel, vergleichbar der Formel  $\alpha$  im Tupel- oder Domänenkalkül [vgl. DB:V-69, DB:V-84]. Ausnahmen bzw. Ergänzungen sind u. a.:
  - die Junktoren heißen `and`, `or`, `not`
  - es gibt mengenwertige Operanden; die Operatoren hierfür sind `in`, `exists`, `any`
  - der Allquantor ist nicht zugelassen
  - ein Operand kann eine Unterabfrage, also wiederum ein komplexer SFW-Block sein
- Der „=“-Operator realisiert einen Verbund (*Join*) zwischen den Relationen der beteiligten Attribute. Mit mehreren, durch `and` verknüpften Gleichheitsbedingungen lassen sich Mehrwege-Joins spezifizieren.
- Es gibt Operatoren für Bereichsselektion, `between`, und zum Mustervergleich, `like`.

# SQL als Datenanfragesprache

## Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *  
from Mitarbeiter  
where ChefPersNr < 8000
```

~>

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5

# SQL als Datenanfragesprache

## Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *  
from Mitarbeiter  
where ChefPersNr < 8000
```

~>

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5

```
select *  
from Mitarbeiter  
where Wohnort like '%öl%'
```

~>

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Wong	3334	Köln	8886	5

# SQL als Datenanfragesprache

## Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *  
from Mitarbeiter as m, ArbeitetInProjekt as a  
where m.PersNr = a.PersNr or ChefPersNr = 9876
```

# SQL als Datenanfragesprache

## Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

**select** \*

**from** Mitarbeiter as m, ArbeitetInProjekt as a

**where** m.PersNr = a.PersNr or ChefPersNr = 9876

↪

Name	PersNr	Wohnort	ChefPersNr	AbtNr	PersNr	ProjektNr
Smith	1234	Weimar	3334	5	1234	1
Smith	1234	Weimar	3334	5	1234	2
Zelaya	9998	Erfurt	9876	4	1234	1
Zelaya	9998	Erfurt	9876	4	1234	2
Zelaya	9998	Erfurt	9876	4	6668	3
Zelaya	9998	Erfurt	9876	4	4534	1

# SQL als Datenanfragesprache

## Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

## Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

# SQL als Datenanfragesprache

## Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

## Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

```
select m1.Name
from Mitarbeiter as m1, Mitarbeiter as m2
where m2.Name = 'Smith' and
       m1.AbtNr = m2.AbtNr and
       m1.Name != 'Smith';
```

# SQL als Datenanfragesprache

## Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

## Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

```
select m1.Name
from Mitarbeiter as m1, Mitarbeiter as m2
where m2.Name = 'Smith' and
       m1.AbtNr = m2.AbtNr and
       m1.Name != 'Smith';
```

```
select m1.Name
from Mitarbeiter as m1, Variante mit Subquery in From-Klausel
       (select * from Mitarbeiter where name = 'Smith') as m2
where m1.AbtNr = m2.AbtNr and
       m1.Name != 'Smith';
```

# SQL als Datenanfragesprache

## Bezug zur Relationenalgebra

Seien  $r_1, r_2$  Relationen über den Schemata  $\mathcal{R}_1 = \{A_1, A_2\}$  bzw.  $\mathcal{R}_2 = \{A_2, A_3\}$ .

```
select  A1, A3  
from    r1 as r3, r2  
where  r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1, A_3} \sigma_{r_3.A_2=r_2.A_2} (\rho_{r_3}(r_1) \times r_2)$$

- ❑ SQL-Select entspricht der Projektion  $\pi$
- ❑ SQL-Alias-Deklaration entspricht der Umbenennung  $\rho$
- ❑ SQL-From entspricht dem kartesischen Produkt  $\times$
- ❑ SQL-Where entspricht der Selektion  $\sigma$

# SQL als Datenanfragesprache

## Bezug zum Tupelkalkül

Seien  $r_1, r_2$  Relationen über den Schemata  $\mathcal{R}_1 = \{A_1, A_2\}$  bzw.  $\mathcal{R}_2 = \{A_2, A_3\}$ .

```
select A1, A3  
from r1 as r3, r2  
where r3.A2 = r2.A2
```

- SQL-Select entspricht der Tupelsynthese auf Basis der freien Variablen:

$$\{(t_1.A_1, t_2.A_3) \mid \dots\}$$

- SQL-From entspricht der Bindung von freien Variablen an Relationen:

$$\{(t_1.A_1, t_2.A_3) \mid r_3(t_1) \wedge r_2(t_2) \dots\}$$

- SQL-Where entspricht einem als Formel spezifizierten Constraint:

$$\{(t_1.A_1, t_2.A_3) \mid r_3(t_1) \wedge r_2(t_2) \wedge t_1.A_2 = t_2.A_2\}$$

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Drei Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select**  $A_1, A_2, \dots, A_n$   
**from**  $r_1, r_2, \dots, r_m$   
**where** `[not] exists`  
**(select... from... where...)**

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Drei Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select**  $A_1, A_2, \dots, A_n$   
**from**  $r_1, r_2, \dots, r_m$   
**where** [not] exists  
    (**select**... **from**... **where**...)
2. **select**  $A_1, A_2, \dots, A_n$   
**from**  $r_1, r_2, \dots, r_m$   
**where**  $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \dots)\}$  [not] in  
    (**select**... **from**... **where**...)
3. **select**  $A_1, A_2, \dots, A_n$   
**from**  $r_1, r_2, \dots, r_m$   
**where**  $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \dots)\}$  {=|<>|<|...} [any | all]  
    (**select**... **from**... **where**...)

## Bemerkungen:

- ❑ Geschachtelte Anfragen heißen auch Unterabfragen bzw. Subqueries.
- ❑ Verwendungsform 1. Das Ergebnis der Subquery wird daraufhin getestet, ob es einen oder ob es keinen Match gibt.
- ❑ Verwendungsform 2. Das Ergebnis der Subquery wird daraufhin getestet, ob es einen bestimmten Attributwert oder ein Tupel enthält.
- ❑ Verwendungsform 3. Ohne `any` bzw. `all`: Das Ergebnis der Subquery muss genau *ein* Element zurückliefern, das dann bzgl. der angegebenen Relation getestet wird. Mit `any` bzw. `all`: Das Ergebnis der Subquery kann eine Menge sein.
- ❑ `=any` ist äquivalent zu `in`, `not in` ist äquivalent zu `<>all`.
- ❑ Subqueries in der Where-Klausel (*Expression Subqueries*) dienen zur Formulierung von Bedingungen. Subqueries in der From-Klausel (*Derived Table Subqueries*) dienen zur Bildung spezialisierter Tabellen für kartesische Produkte.
- ❑ Subqueries können weiter geschachtelt sein, also ihrerseits Subqueries enthalten.
- ❑ In Subqueries kann auf Relationen der sie umschließenden Umgebung Bezug genommen werden. Stichwort: korrelierte Unterabfragen (*Correlated Subqueries*)
- ❑ Abhängig von der Strategie bzw. Güte der Anfrageoptimierung des DBMS ergeben sich bei semantisch äquivalenten Anfrageformulierungen stark unterschiedliche Antwortzeiten.
- ❑ Subqueries erhöhen nicht die Ausdruckskraft von SQL, sondern erleichtern nur die Formulierung. Die Semantik jeder Subquery lässt sich mit entsprechenden Join-Operationen nachbilden.

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

## Anfrage

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

## Anfrage

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

## Relationenalgebra

$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='Bremen'}(\text{Teilnehmer}))$

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

## Anfrage

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

## Relationenalgebra

$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='Bremen'}(\text{Teilnehmer}))$

## SQL Variante (a)

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt, Teilnehmer t
where nt.TeilnNr = t.TeilnNr and t.Ort = 'Bremen'
```

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

## Anfrage

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

SQL Variante (b), korrelierte Unterabfrage

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where exists
    (select *
     from Teilnehmer t
     where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

# SQL als Datenanfragesprache

## Geschachtelte Anfragen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

## Anfrage

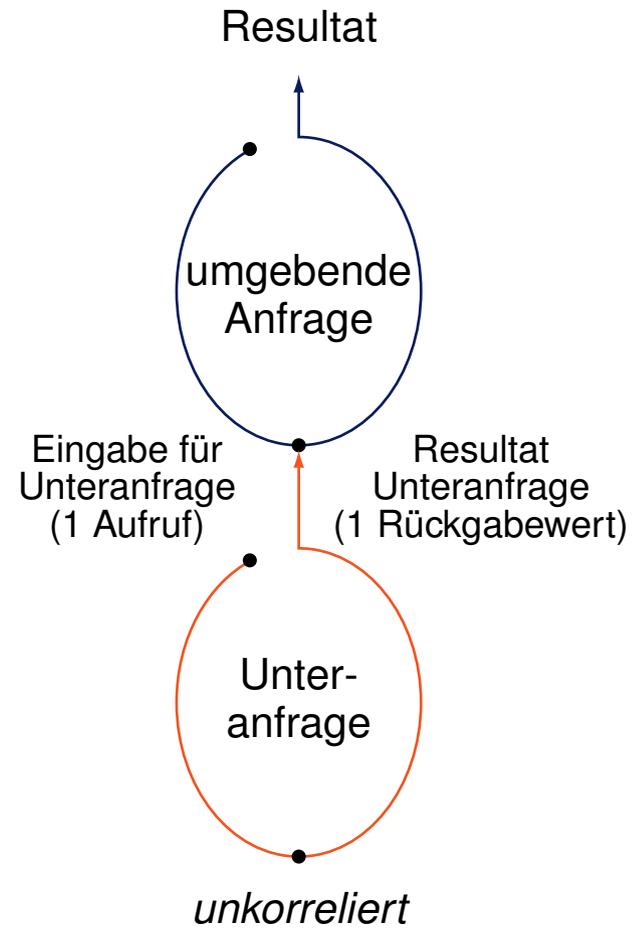
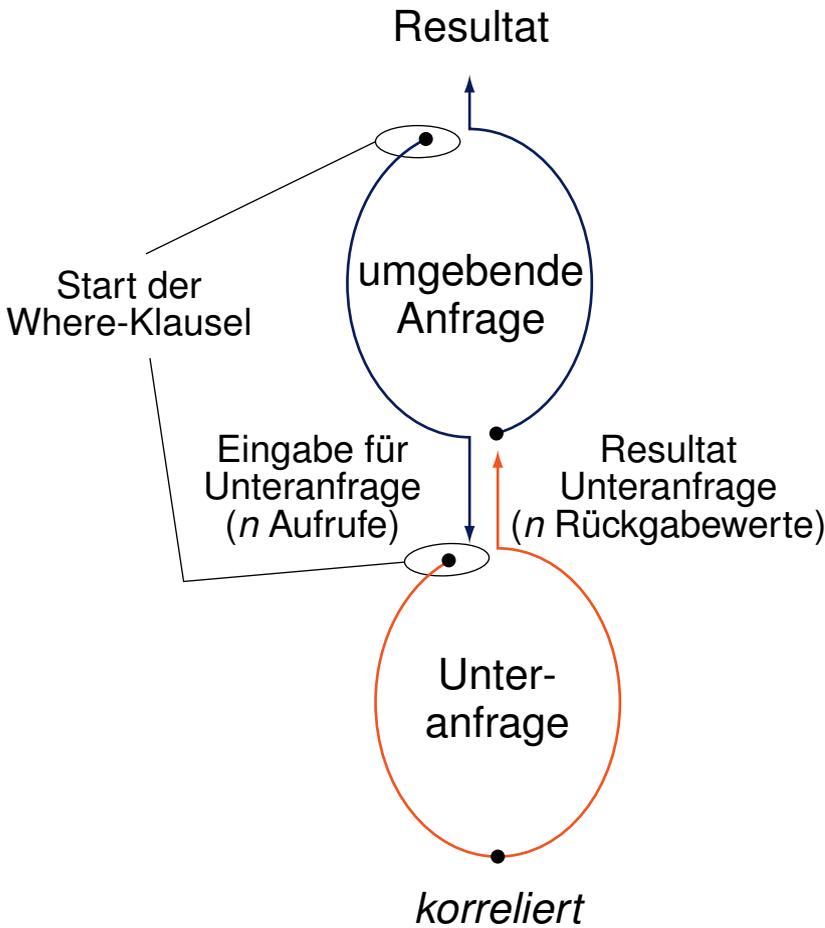
„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

SQL Variante (c), unkorrelierte Unterabfrage

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where nt.TeilnNr in
    (select TeilnNr
     from Teilnehmer
     where Ort = 'Bremen')
```

# SQL als Datenanfragesprache

## Geschachtelte Anfragen



[vgl. Scholl IS, 2002/03]

# SQL als Datenanfragesprache

## Allquantifizierung

- ❑ SQL-89 und SQL-92 besitzen keinen Allquantor.
- ❑ Eine Allquantifizierung muss durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden.
- ❑ Alternativ kann eine Allquantifizierung auch mit der Aggregatfunktion `count` nachgebildet werden.

# SQL als Datenanfragesprache

## Allquantifizierung

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

## Anfrage

„Wer nimmt an allen Kursen teil?“

# SQL als Datenanfragesprache

## Allquantifizierung

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

## Anfrage

„Wer nimmt an allen Kursen teil?“

## Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div \pi_{\text{KursNr}}(\text{Kurs})))$

# SQL als Datenanfragesprache

## Allquantifizierung

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

## Anfrage

„Wer nimmt an allen Kursen teil?“

## Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div \pi_{\text{KursNr}}(\text{Kurs})))$

## Tupelkalkül

$\{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \forall t_3(\neg \text{Kurs}(t_3) \vee \exists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr}))\}$

$\approx \{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \exists t_3(\neg(\neg \text{Kurs}(t_3) \vee \exists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})))\}$

$\approx \{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \exists t_3(\text{Kurs}(t_3) \wedge \exists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr}))\}$

# SQL als Datenanfragesprache

## Allquantifizierung

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

## Anfrage

„Wer nimmt an allen Kursen teil?“

## Tupelkalkül

$$\{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \exists t_3(\text{Kurs}(t_3) \wedge \exists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr}))\}$$

## SQL

```
select Name
from Teilnehmer t1
where not exists
  (select *
   from Kurs t3
   where not exists
     (select *
      from nimmt_teil t2
      where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

## Bemerkungen:

- Natürlichsprachliche Formulierung der SQL-Anfrage:  
„Liefere alle Teilnehmer, bei denen kein Kurs existiert, an dem keiner teilnimmt.“
- Bei (formalen, logischen, natürlichen) Sprachen unterscheidet man zwischen Objektsprache und Metasprache. Die Objektsprache dient zum Formulieren von Sätzen, um *mit* der Sprache zu kommunizieren. Die Metasprache dient zur Formulierung von Zusammenhängen *über* die Objektsprache.
- Die Formelbezeichner  $\alpha$ ,  $\beta$  und  $\gamma$ , die Prädikatsbezeichner  $P$  und  $Q$ , die Quantoren  $\forall$  und  $\exists$ , die Variablenbezeichner sowie die Junktoren  $\wedge$ ,  $\vee$  und  $\neg$  gehören zur Objektsprache. Das  $\approx$ -Zeichen ist ein Zeichen der Metasprache und steht für „ist logisch äquivalent mit“.
- Es gelten folgende Zusammenhänge:

$$\forall x P(x) \approx \neg \exists x (\neg P(x))$$

$$\alpha \rightarrow \beta \approx \neg \alpha \vee \beta$$

$$\neg(\alpha \vee \beta) \approx \neg \alpha \wedge \neg \beta$$

# SQL als Datenanfragesprache

## Mengenoperationen

Seien  $r_1, r_2$  Relationen über den Schemata  $\mathcal{R}_1 = \{A_1, A_2\}$  bzw.  $\mathcal{R}_2 = \{A_2, A_3\}$ . Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

□ Vereinigung.

```
select A2 from r1 where <condition1>  
union [all]  
select A2 from r2 where <condition2>
```

# SQL als Datenanfragesprache

## Mengenoperationen

Seien  $r_1, r_2$  Relationen über den Schemata  $\mathcal{R}_1 = \{A_1, A_2\}$  bzw.  $\mathcal{R}_2 = \{A_2, A_3\}$ . Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

- Vereinigung.

```
select A2 from r1 where <condition1>  
union [all]  
select A2 from r2 where <condition2>
```

- Durchschnitt.

```
select A2 from r1, r2  
where r1.A2 = r2.A2 and <condition1> and <condition2>
```

# SQL als Datenanfragesprache

## Mengenoperationen

Seien  $r_1, r_2$  Relationen über den Schemata  $\mathcal{R}_1 = \{A_1, A_2\}$  bzw.  $\mathcal{R}_2 = \{A_2, A_3\}$ . Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

- Vereinigung.

```
select A2 from r1 where <condition1>  
union [all]  
select A2 from r2 where <condition2>
```

- Durchschnitt.

```
select A2 from r1, r2  
where r1.A2 = r2.A2 and <condition1> and <condition2>
```

- Differenz.

```
select A2 from r1  
where <condition1> and r1.A2 not in  
    (select A2 from r2  
     where <condition2>)
```

## Bemerkungen:

- ❑ In der Relationenalgebra sind Mengenoperationen nur über Relationen mit dem gleichen Relationenschema zugelassen.
- ❑ In SQL spielen im Zusammenhang mit Mengenoperationen die Namen der Attribute keine Rolle: Mengenoperationen erfordern nur, dass die Listen der Attribute der beteiligten Relationen positionsweise kompatibel sind.
- ❑ Zwei Attribute sind kompatibel zueinander, falls sie kompatible Wertebereiche haben. Das heißt, dass die Wertebereiche entweder
  1. gleich sind oder
  2. beide auf dem Typ „Character“ basieren oder
  3. beide von einem numerischen Typ sind.

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Aggregatfunktionen, auch built-in-Funktionen genannt, führen Operationen auf *Tupelmengen* durch und „verdichten“ so eine Menge von Werten zu einem einzelnen Wert.

### Aggregatfunktionen in SQL-89:

Name	Beschreibung
<code>count(*)</code>	Anzahl der Tupel
<code>count([distinct] &lt;attribute&gt;)</code>	Anzahl der Attributwerte
<code>max(&lt;attribute&gt;)</code>	Maximum der Attributwerte
<code>min(&lt;attribute&gt;)</code>	Minimum der Attributwerte
<code>avg([distinct] &lt;attribute&gt;)</code>	Durchschnitt der Attributwerte
<code>sum([distinct] &lt;attribute&gt;)</code>	Summe der Attributwerte

## Bemerkungen:

- ❑ Aggregatfunktionen können nicht geschachtelt werden.

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)  
from Teilnehmer
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)  
from Teilnehmer
```

„Wieviele Teilnehmer kommen aus Hamburg?“

```
select count(*)  
from Teilnehmer  
where Ort = 'Hamburg'
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)  
from Teilnehmer
```

„Wieviele Teilnehmer kommen aus Hamburg?“

```
select count(*)  
from Teilnehmer  
where Ort = 'Hamburg'
```

„Wie ist das Durchschnittsalter der Kursleiter?“

```
select avg(Alter)  
from Kursleiter
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where Alter =
    (select max(Alter)
     from Kursleiter)
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where Alter =
    (select max(Alter)
     from Kursleiter)
```

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 <
    (select count(*)
     from Teilnehmer t
     where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

## Anfragen

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where Alter =
    (select max(Alter)
     from Kursleiter)
```

„Liefere alle Kursangebote, für die Teilnehmer aus Bremen gebucht haben.“

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 < exists // vgl. Variante (b)
    (select count(*) * // vgl. Variante (b)
     from Teilnehmer t
     where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

```
select ...  
from ...  
[where ...]  
[group by <attribute1>, <attribute2>, ...]  
[having <condition>]
```

- Die Group-By-Klausel bewirkt eine *Teilmenge*nbildung: Alle Tupel, die gleiche Werte in der spezifizierten Attributliste haben, werden als eine Teilmenge betrachtet.
- In der Select-Klausel können nur Aggregatfunktionen oder Attribute, nach denen gruppiert wurde, vorkommen.
- Aggregatfunktionen werden auf die *Teilmengen* angewandt.
- Mittels der Having-Klausel lassen sich Nebenbedingungen für die *Teilmengen* formulieren.

## Bemerkungen:

- ❑ Unterschied zwischen der Where-Klausel und der Having-Klausel: Mit der Where-Klausel werden Tupel eliminiert, mit der Having-Klausel werden Gruppen eliminiert.
- ❑ *Logische* Ausführungsreihenfolge: from → where → group by → having → select

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

### Anfrage

„Für welche Programmierfähigkeiten gibt es mehrere Mitarbeiter?“

```
select count(*) as Anzahl, Skill as Faehigkeit
from MitarbeiterSkill
group by Skill
having Anzahl > 1
```

~>

Anzahl	Faehigkeit
2	Java
2	Linux

# SQL als Datenanfragesprache

## Aggregat- und Gruppierungsfunktionen

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

### Anfrage

„Wie ist die Verteilung der Fähigkeiten in Abteilung 5?“

```
select count(*) as Anzahl, Skill as Faehigkeit
from Mitarbeiter m, MitarbeiterSkill s
where AbtNr = 5 and m.PersNr = s.PersNr
group by Skill
```

~>

Anzahl	Faehigkeit
2	Java
1	C++
1	Linux

# SQL als Datenanfragesprache

## Nullwerte

Der spezielle Wert „Null“,  $\perp$ , ist als Wert in jedem Datentyp vorhanden. Folgende Ursachen können der Verwendung von Null als Wert zugrunde liegen:

1. Der Wert des Attributes ist unbekannt.
2. Der Wert des Attributes ist bekannt, soll aber nicht gespeichert werden.
3. Im Wertebereich des Attributes ist kein adäquater Wert vorhanden.

# SQL als Datenanfragesprache

## Nullwerte

Der spezielle Wert „Null“,  $\perp$ , ist als Wert in jedem Datentyp vorhanden. Folgende Ursachen können der Verwendung von Null als Wert zugrunde liegen:

1. Der Wert des Attributes ist unbekannt.
2. Der Wert des Attributes ist bekannt, soll aber nicht gespeichert werden.
3. Im Wertebereich des Attributes ist kein adäquater Wert vorhanden.

## Vergleiche auf Null

- Vergleichsoperatoren liefern Null, falls einer der Operanden Null ist.
- Für den Vergleich mit Null dienen die Operatoren `is` und `is not`.

Beispiele		Wert
<code>null is null,</code>	<code>3 &lt; 4</code>	<code>true</code>
<code>null is not null,</code>	<code>4 &lt; 3</code>	<code>false</code>
<code>null = null,</code>	<code>4 &lt; null</code>	<code>null</code>

# SQL als Datenanfragesprache

## Nullwerte

Mit Hilfe von Null ist in SQL eine dreiwertige Logik realisiert.

Boolsche Semantik:

$\neg$	
true	false
unknown	unknown
false	true

SQL Werte:

<b>not</b>	
1	0
null	null
0	1

# SQL als Datenanfragesprache

## Nullwerte

Mit Hilfe von Null ist in SQL eine dreiwertige Logik realisiert.

### Boolsche Semantik:

$\neg$	
true	false
unknown	unknown
false	true

$\wedge$	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

$\vee$	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

### SQL Werte:

not	
1	0
null	null
0	1

and	1	null	0
1	1	null	0
null	null	null	0
0	0	0	0

or	1	null	0
1	1	1	1
null	1	null	null
0	1	null	0

## Bemerkungen:

- ❑ In arithmetischen Ausdrücken werden Nullwerte propagiert: ist ein Operand einer Operation Null, so wird auch das Ergebnis Null.
- ❑ In einer Where-Klausel werden nur Tupel berücksichtigt, die zu `true` evaluieren. Das heißt, Tupel, die zu `unknown` evaluieren, werden nicht in das Ergebnis aufgenommen.
- ❑ Bei einer Gruppierung wird `null` als eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.

# SQL als Datenanfragesprache

## Zusammengesetzte Terme

In Select- und Where-Klauseln können an der Stelle von Attributen auch zusammengesetzte Terme wie arithmetische Ausdrücke oder Ausdrücke, die Zeichenketten manipulieren, stehen.

Kursleiter		
PersNr	Name	Alter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

```
select PersNr*Alter as Glueckszahl  
from Kursleiter
```

~→

Glueckszahl
438009
996912
1622871

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

### □ SQL-89:

```
select *  
from Mitarbeiter, MitarbeiterSkill
```

```
select *  
from Mitarbeiter, MitarbeiterSkill  
where Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

### □ SQL-89:

```
select *  
from Mitarbeiter, MitarbeiterSkill
```

```
select *  
from Mitarbeiter, MitarbeiterSkill  
where Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

### □ SQL-92:

```
select *  
from Mitarbeiter cross join MitarbeiterSkill
```

```
select *  
from Mitarbeiter join MitarbeiterSkill  
    on Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

- Gleichverbund (Equi-Join) in SQL-92:

```
select *
```

```
from Mitarbeiter join MitarbeiterSkill
```

```
    using (PersNr)
```

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

- Gleichverbund (Equi-Join) in SQL-92:

```
select *  
from Mitarbeiter join MitarbeiterSkill  
    using (PersNr)
```

- natürlicher Verbund in SQL-92:

```
select *  
from Mitarbeiter natural join MitarbeiterSkill
```

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

- Gleichverbund (Equi-Join) in SQL-92:

```
select *  
from Mitarbeiter join MitarbeiterSkill  
    using (PersNr)
```

- natürlicher Verbund in SQL-92:

```
select *  
from Mitarbeiter natural join MitarbeiterSkill
```

- äußere Verbunde in SQL-92:

```
[natural] [left | right] outer join  
    using (...)  
    on ...
```

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92: Joins

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Köln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

### Anfrage

„Wie ist die Personalnummer vom Chef, der keine Ahnung hat?“

```
select distinct ChefPersNr
from Mitarbeiter m left outer join Mitarbeiterskill s
    on m.ChefPersNr = s.PersNr
where Skill is null
```

~>

ChefPersNr
8886

# SQL als Datenanfragesprache

## SQL-89 versus SQL-92

- Es gibt die Mengenoperationen `union`, `intersect` und `except`, für die mittels `corresponding` noch eine Attributliste vereinbart werden kann.
- In der For-Klausel können mittels eines `Select-From-Where-Blocks` virtuelle Relationen erzeugt und benannt werden.
- In Bedingungen der `Where-Klausel` lassen sich nicht nur skalare Ausdrücke, sondern auch Tupel spezifizieren.

# Kapitel DB: VI (Fortsetzung)

- I. Einführung und grundlegende Konzepte von Datenbanken
- II. Datenbankentwurf und Datenbankmodelle
- III. Konzeptueller Datenbankentwurf
- IV. Logischer Datenbankentwurf mit dem relationalen Modell
- V. Grundlagen relationaler Anfragesprachen

## VI. Die relationale Datenbanksprache SQL

- ❑ Einführung
- ❑ SQL als Datenanfragesprache
- ❑ SQL als Datendefinitionssprache
- ❑ SQL als Datenmanipulationssprache
- ❑ Sichten
- ❑ SQL vom Programm aus

## VII. Entwurfstheorie relationaler Datenbanken

# SQL als Datendefinitionssprache

## Datentypen

Die wichtigsten Datentypen für die Wertebereiche von Attributen sind Zahlen, Zeichenketten und Datumsangaben.

---

Typ	Beschreibung
char(n)	Zeichenstring mit fester Länge n. Synonym: character(n)
varchar(n)	Zeichenstring mit variabler aber maximaler Länge n. Synonyme: char varying(n), character varying(n)
int	Wert einer maschinenabhängigen, endlichen Teilmenge der ganzen Zahlen. Synonym: integer
smallint	eine maschinenabhängige Teilmenge des int-Wertebereichs
numeric(z, n)	Fixpunktzahl (Dezimalzahl) mit spezifizierbarer Genauigkeit, z = Gesamtanzahl der Ziffern, n = Anzahl der Nachkommastellen. Synonym: decimal(z, n)
real	Fließkommazahl mit maschinenabhängiger Genauigkeit

---

# SQL als Datendefinitionssprache

## Datentypen (Fortsetzung)

---

Typ	Beschreibung
double precision	Fließkommazahl mit doppelter, maschinenabhängiger Genauigkeit.
float(n)	Fließkommazahl mit spezifizierbarer Genauigkeit von $\geq n$ Stellen.
bit(n)	Bitstring mit fester Länge n.
bit varying(n)	Bitstring mit variabler aber maximaler Länge n.
blob	Binary Large Object. Variabel lange Byte-Sequenz von maximal 4 GB. Zum Speichern von Videosequenzen, Bilder, Audio-Dateien, etc.
date	Kalenderdatum mit Jahr (4 Stellen), Monat (2 Stellen), Tag (2 Stellen). Format: JJJJ-MM-TT
time	Tageszeit in Stunden, Minuten und Sekunden, Format: HH:MM:SS
time with time zone	Zeitunterschied zu GMT (6 Stellen), Bereich: +13:00 bis - 12:59
timestamp	Wert, der Datum und Tageszeit enthält.
interval	Wert, um den ein absoluter Wert vom Typ date, time oder timestamp in/dekrementiert werden kann. Dient zur Formulierung von Jahr/Monat- oder Tag/Zeit-Intervallen.

---

# SQL als Datendefinitionssprache

## Domains (SQL-92)

```
create {domain | datatype} [as] <domain> <datatype>
[[not] null]
[default value]
[check (<condition>)]
```

- ❑ `<datatype>` bezeichnet einen integrierten Datentyp mit Gesamtstellenzahl und Dezimalstellen.
- ❑ Die Deklaration von Domains ist vergleichbar mit einfachen, nicht-geschachtelten Typ-Vereinbarungen in einer Programmiersprache.
- ❑ Domains können in Create-Table-Anweisungen für Attributdeklarationen verwendet werden.

# SQL als Datendefinitionssprache

## Domains (SQL-92)

```
create {domain | datatype} [as] <domain> <datatype>
[[not] null]
[default value]
[check (<condition>)]
```

- ❑ <datatype> bezeichnet einen integrierten Datentyp mit Gesamtstellenzahl und Dezimalstellen.
- ❑ Die Deklaration von Domains ist vergleichbar mit einfachen, nicht-geschachtelten Typ-Vereinbarungen in einer Programmiersprache.
- ❑ Domains können in Create-Table-Anweisungen für Attributdeklarationen verwendet werden.

## Beispiele:

```
create domain address char(35) null
```

```
create domain Stadtstaaten varchar(10) default 'Berlin'
```

# SQL als Datendefinitionssprache

## Check-Klausel

`check (<condition>)`

- Die Check-Klausel dient zur Festlegung lokaler Integritätsbedingungen in Domains, Attributen und Relationenschemata.
- Die Check-Klausel erlaubt die Formulierung von Prädikaten in der Art, wie sie auch in der Where-Klausel bei Anfrageoperationen spezifiziert werden können.

# SQL als Datendefinitionssprache

## Check-Klausel

`check (<condition>)`

- ❑ Die Check-Klausel dient zur Festlegung lokaler Integritätsbedingungen in Domains, Attributen und Relationenschemata.
- ❑ Die Check-Klausel erlaubt die Formulierung von Prädikaten in der Art, wie sie auch in der Where-Klausel bei Anfrageoperationen spezifiziert werden können.

## Beispiele:

```
check(Alter between 18 and 27)
```

```
check(Dauer > 3)
```

```
create domain Stadtstaaten varchar(10) default 'Berlin'  
check (value in ('Berlin', 'Hamburg', 'Bremen'))
```

```
check((select avg(Alter) from Reiseleiter) <  
      (select sum(TeilnAnz) from Reisen where Stadt='Paris'))
```

# SQL als Datendefinitionssprache

## Datenbanken

```
create database <database>
```

```
[with {[buffered] log | log mode ansi}]
```

- ❑ Erstellt eine Datenbank und macht diese zur aktuellen Datenbank. Mit dem Erzeugen einer Datenbank werden auch die Systemrelationen (Data Dictionary) angelegt. Der Datenbankname muss eindeutig sein.
- ❑ Durch Angabe der with-Option wird die Protokollierung von Transaktionen aktiviert. Diese wird im Falle eines Datenbankfehlers zum Wiederherstellen der Daten benötigt. Gepufferte Protokollierung erhöht dabei die Performance der Protokollierung.

```
drop database <database>
```

- ❑ Löscht eine Datenbank einschließlich aller Daten, Indizes und Systemrelationen. Es kann weder die aktuelle Datenbank noch eine Datenbank, die gerade von anderen Benutzern verwendet wird, gelöscht werden.

```
close database <database>
```

- ❑ Schließt die aktuell geöffnete Datenbank. Eine geschlossene Datenbank kann gelöscht werden.

# SQL als Datendefinitionssprache

## Relationen definieren

```
create table <table>
(<attribute1> <datatype> [default <value>] [not null] [<attr_int_cond>],
 <attribute2> <datatype> [default <value>] [not null] [<attr_int_cond>],
 ...
 [<rel_int_cond>]
)
```

```
<attr_int_cond> ::= {unique | primary key |
                    references <table>[(<attribute>)]}
```

```
<rel_int_cond> ::= {unique (<attribute1>, ...) |
                    primary key (<attribute1>, ...) |
                    check (<condition>) |
                    foreign key (<attribute1>, ...) references
                    <table>[(<attribute1>, ...)] [<action>]}
```

## Bemerkungen:

- ❑ Mittels `default` kann ein Standardwert für Attribute vorgegeben werden.
- ❑ Integritätsbedingungen beziehen sich auf ein Tupel, eine Relation oder mehrere Relationen.
- ❑ Besteht ein Schlüssel aus *einem* Attribut, kann die entsprechende Integritätsbedingung mit `<attr_int_cond>` direkt bei der Attributdeklaration vereinbart werden.
- ❑ Integritätsbedingungen für Schlüssel, die aus mehreren Attributen bestehen, werden mittels `<rel_int_cond>` vereinbart.
- ❑ Mit `<action>` lassen sich Maßnahmen definieren, um die Fremdschlüsselbeziehungen in der Datenbank aufrechtzuerhalten. Jedes Mal, wenn ein Primärschlüsselwert geändert oder aus der Datenbank gelöscht wird, kann es entsprechende Fremdschlüsselwerte in anderen Relationen geben, die auf die eine oder andere Art geändert zu ändern sind. Hierfür kann entweder eine On-Update-Klausel, eine On-Delete-Klausel oder auch beide angegeben werden, gefolgt von einer der folgenden Aktionstypen: `cascade`, `set null`, `set default`, `restrict`.

# SQL als Datendefinitionssprache

## Relationen definieren

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

## Verwendung der Form `<attr_int_cond>`:

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

## Verwendung der Form `<rel_int_cond>`:

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

## Bemerkungen:

- In SQL-92 sind Primary-Key-Attribute implizit „unique“ und „not null“.

# SQL als Datendefinitionssprache

## Relationen definieren

```
create table Buch_Versionen
(ISBN char(20),
 Auflage smallint check (Auflage > 0),
 Jahr integer check (Jahr between 1800 and 2020),
 Seitenzahl integer check (Seiten > 0),
 Preis decimal(8,2) check (Preis ≤ 250),

primary key (ISBN, Auflage),
foreign key (ISBN) references Buecher (ISBN),

check ((select sum(Preis) from Buch_Versionen) <
      (select sum(Budget) from Lehrstuehle))

)
```

# SQL als Datendefinitionssprache

## Relationen ändern

```
alter table <table>
{add <attribute> <datatype> ...[before <attribute>] |
 modify <attribute> <datatype> |
 drop <attribute> |
 <add_cons>}
```

```
<add_cons>::={add constraint {unique | primary key}
              (<attributel>, ...) |
              add constraint foreign key (<attributel>, ...)
              references <table> (<attributel>, ...) |
              add constraint check (<condition>)}
```

- Modifiziert eine bestehende Relation. Es können Attribute eingefügt oder gelöscht, Constraints hinzugefügt, verändert oder gelöscht werden.

# SQL als Datendefinitionssprache

## Relationen löschen

```
drop table <table>
```

- ❑ Löscht alle in einer Relation enthaltenen Daten, die Indizes und Constraints auf den Attributen inklusive aller Referenz-Constraints, alle mit der Relation verbundenen Synonyme sowie alle für diese Relation vergebenen Berechtigungen.
- ❑ Es können nur Relationen der aktuellen Datenbank gelöscht werden.
- ❑ Das Löschen von Relationen mit Systeminformation ist mit dieser Anweisung nicht möglich.

# SQL als Datendefinitionssprache

## Datendefinitionsbefehle im Überblick

---

	database	table	index	view	synonym	schema
create	•	•	•	•	•	•
alter		•	•			
drop	•	•	•	•	•	
close	•					

---

# SQL als Datenmanipulationssprache

## Einfügen von Tupeln

```
insert into <table>  
[(<attribute1>,...)]  
{values (<expression1>,...) {, (...)}0 | <sfw-block>}
```

- Einfügen von vollständigen oder unvollständigen Tupeln. Eine *Menge* von Tupeln kann mittels eines Select-From-Where-Blocks spezifiziert und in einem Schritt eingefügt werden.
- Alle einzufügenden Werte müssen die Integritätsbedingungen erfüllen.

## Zwei Verwendungsformen:

1. Ohne Attributnamen: Anzahl, Reihenfolge und Datentyp der Werte müssen der Definition der Relation entsprechen. Die Reihenfolge der Attribute ist in der Relation „Syscolumns“ definiert.
2. Mit Attributnamen: Einfügen der Werte gemäß der Attributnamen. Fehlende Attribute erhalten den Nullwert.

# SQL als Datenmanipulationssprache

## Einfügen von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Gebührenrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Gebuehren values (1, G08, 200, null) bzw.
```

```
insert into Gebuehren (AngNr, KursNr, TnNr) values (1, G08, 200)
```

# SQL als Datenmanipulationssprache

## Einfügen von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Gebührenrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Gebuehren values (1, G08, 200, null) bzw.
```

```
insert into Gebuehren (AngNr, KursNr, TnNr) values (1, G08, 200)
```

„Füge ein neues Kursangebot mit AngNr 3 für G08 für den 15. März 1991 in Ulm ein.“

```
insert into Angebot values (3, G08, 15-03-1991, ULM)
```

# SQL als Datenmanipulationssprache

## Einfügen von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Gebührenrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Gebuehren values (1, G08, 200, null) bzw.
```

```
insert into Gebuehren (AngNr, KursNr, TnNr) values (1, G08, 200)
```

„Füge ein neues Kursangebot mit AngNr 3 für G08 für den 15. März 1991 in Ulm ein.“

```
insert into Angebot values (3, G08, 15-03-1991, ULM)
```

„Die Relation Gebuehren sei leer. Fülle die Attribute AngNr, KursNr und TnNr mit den Einträgen der Relation Nimmt\_teil. Das Attribut Gebuehr soll ohne Wert bleiben.“

```
insert into Gebuehren (AngNr, KursNr, TnNr)
```

```
select *
```

```
from nimmt_teil
```

# SQL als Datenmanipulationssprache

## Löschen von Tupeln

```
delete from <table>  
[where <condition>]
```

- Löscht alle Tupel, die `<condition>` erfüllen, aus `<table>`. Die leere Relation bleibt als Eintrag im Katalog erhalten.

# SQL als Datenmanipulationssprache

## Löschen von Tupeln

„Lösche alle Tupel aus der Teilnehmer-Relation.“

```
delete  
from Teilnehmer
```

„Lösche in der nimmt\_teil-Relation alle Kurse, die vor dem 1. März 1990 stattgefunden haben.“

```
delete  
from nimmt_teil  
where (AngNr, KursNr) in  
      (select AngNr, KursNr  
       from Angebot  
       where Datum < 01-03-1990)
```

# SQL als Datenmanipulationssprache

## Ändern von Tupeln

```
update <table> [[as] <alias>]
set <attribute1> = <expression1>
  [, <attribute2> = <expression2>, ...]
[where <condition>]
```

# SQL als Datenmanipulationssprache

## Ändern von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Std\_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Kursgebühren um 10%.“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Kursgebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

# SQL als Datenmanipulationssprache

## Ändern von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Std\_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Kursgebühren um 10%.“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Kursgebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

„Setze alle Gebühren, für die noch kein Wert spezifiziert wurde, auf die Standardgebühr.“

```
update Gebuehren g
set g.Gebuehr =

where g.Gebuehr is null
```

# SQL als Datenmanipulationssprache

## Ändern von Tupeln

Gegeben seien folgende Relationenschemata:

- Gebuehren = {AngNr, KursNr, TnNr, Gebuehr}
- Std\_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Kursgebühren um 10%.“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Kursgebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Gebuehren
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

„Setze alle Gebühren, für die noch kein Wert spezifiziert wurde, auf die Standardgebühr.“

```
update Gebuehren g
set g.Gebuehr =
    (select s.Gebuehr from Std_Gebuehr s where g.KursNr = s.KursNr)
where g.Gebuehr is null
```

## Bemerkungen:

- Ist ein SFW-Block in einer Gleichung zugewiesen, so darf der SFW-Block nur *ein* Tupel als Return-Wert haben. Siehe Beispiel: `g.Gebuehr = (select ...)`

# Kapitel DB: VI (Fortsetzung)

- I. Einführung und grundlegende Konzepte von Datenbanken
- II. Datenbankentwurf und Datenbankmodelle
- III. Konzeptueller Datenbankentwurf
- IV. Logischer Datenbankentwurf mit dem relationalen Modell
- V. Grundlagen relationaler Anfragesprachen

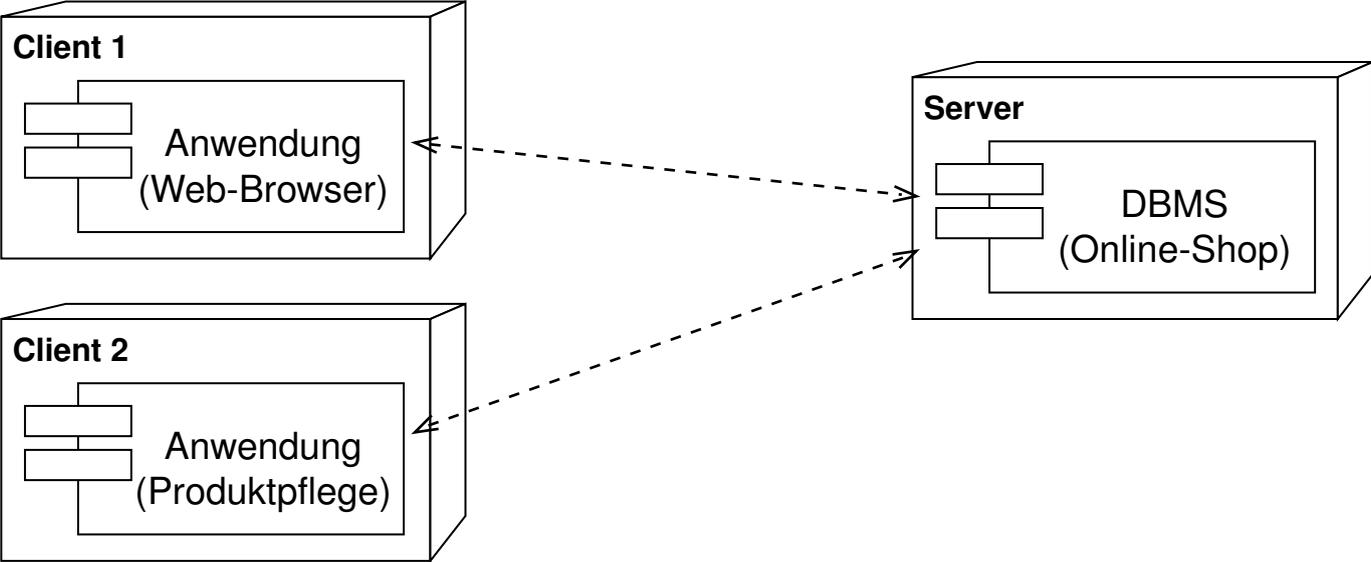
## VI. Die relationale Datenbanksprache SQL

- ❑ Einführung
- ❑ SQL als Datenanfragesprache
- ❑ SQL als Datendefinitionssprache
- ❑ SQL als Datenmanipulationssprache
- ❑ Sichten
- ❑ SQL vom Programm aus

## VII. Entwurfstheorie relationaler Datenbanken

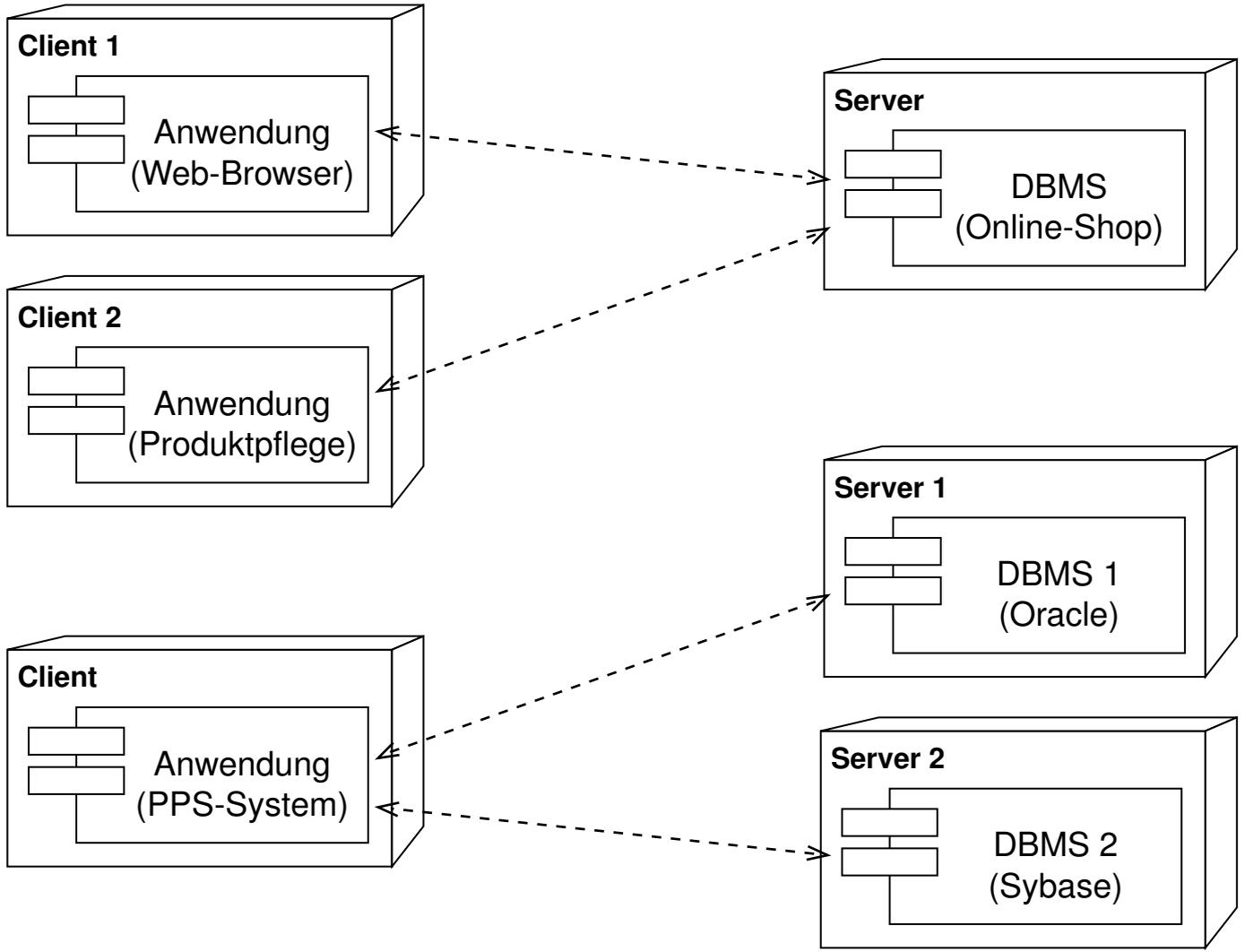
# SQL vom Programm aus

## Anwendungsszenarien



# SQL vom Programm aus

## Anwendungsszenarien



# SQL vom Programm aus

## Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen
2. Einbettung von Datenbanksprachen in Programmiersprachen
  - Vorteil: (SQL-)Statements lassen sich zur Übersetzungszeit überprüfen und optimieren
  - Prinzip: SQL-Anweisungen werden im Programmquelltext ausgezeichnet
  - Beispiel: Embedded SQL; Realisierung für die Programmiersprache Java als SQLJ
  - Erweiterung: Dynamic SQL

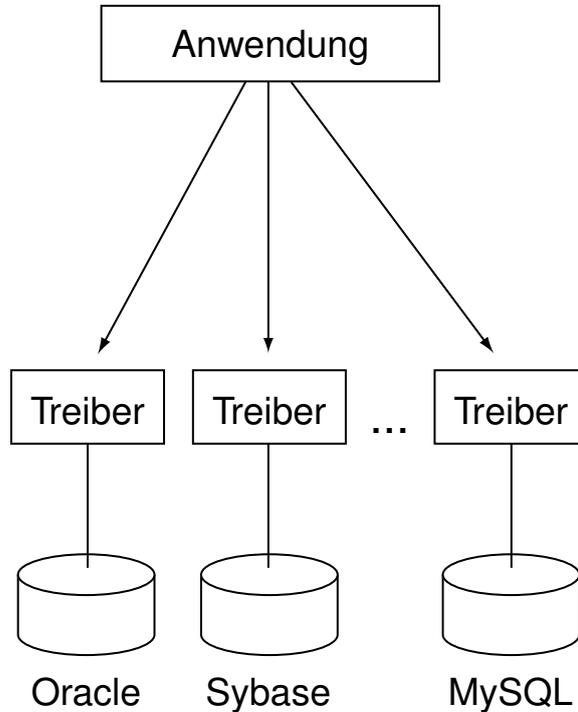
# SQL vom Programm aus

## Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen
2. Einbettung von Datenbanksprachen in Programmiersprachen
  - ❑ Vorteil: (SQL-)Statements lassen sich zur Übersetzungszeit überprüfen und optimieren
  - ❑ Prinzip: SQL-Anweisungen werden im Programm Quelltext ausgezeichnet
  - ❑ Beispiel: Embedded SQL; Realisierung für die Programmiersprache Java als SQLJ
  - ❑ Erweiterung: Dynamic SQL
3. Programmierschnittstelle, Application Programming Interface (API)
  - ❑ Vorteil: hohe Flexibilität
  - ❑ Prinzip: SQL-Anweisungen werden als (dynamisch generierbarer) Text an das Datenbanksystem übergeben

# SQL vom Programm aus

## Programmierschnittstelle

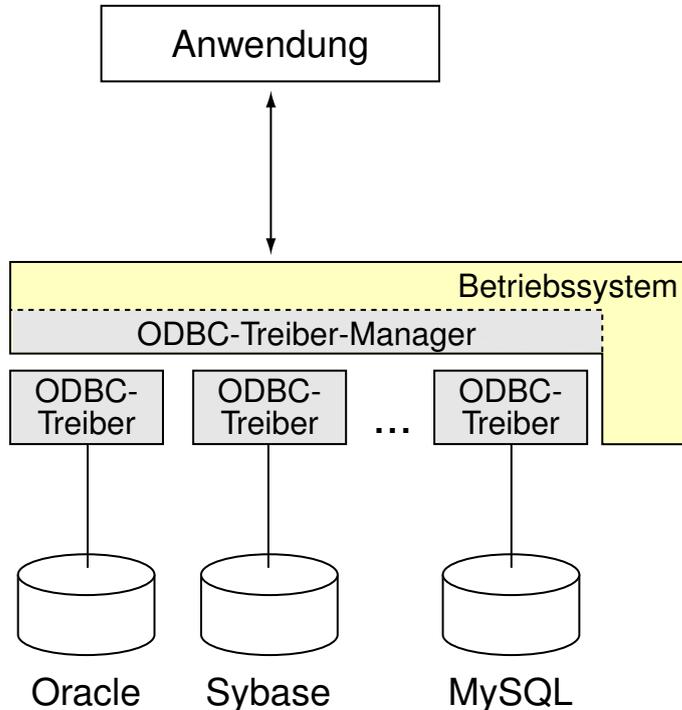


Nachteile hinsichtlich Wartbarkeit:

- ❑ Code für jedes DBMS ist zu entwickeln
- ❑ Datenquellen nur mit Aufwand austauschbar

# SQL vom Programm aus

## Programmierschnittstelle (Fortsetzung)

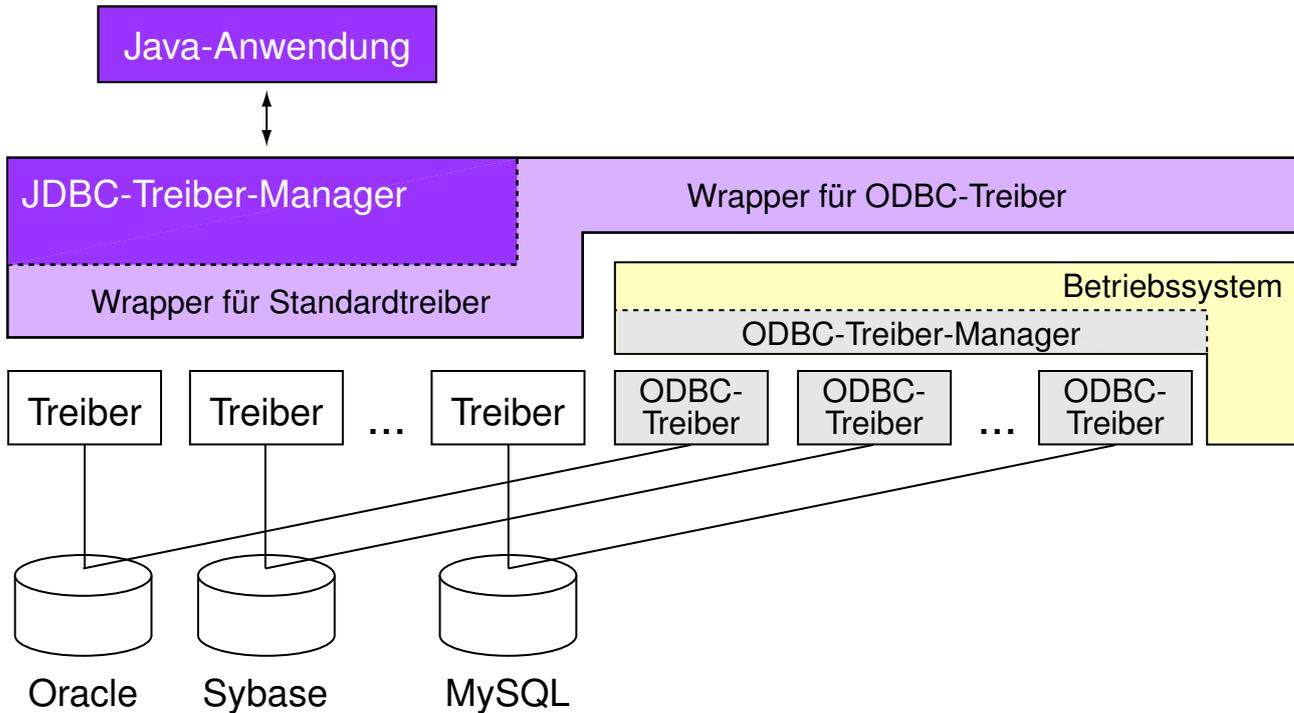


→ Standardisierung der Schnittstelle: Open Database Connectivity, ODBC

Die ODBC-Schnittstelle definiert eine Bibliothek mit Funktionsaufrufen, mit denen eine Verbindung zur Datenquelle hergestellt, SQL-Anweisungen ausgeführt und Ergebnisse abgerufen werden können.

# SQL vom Programm aus

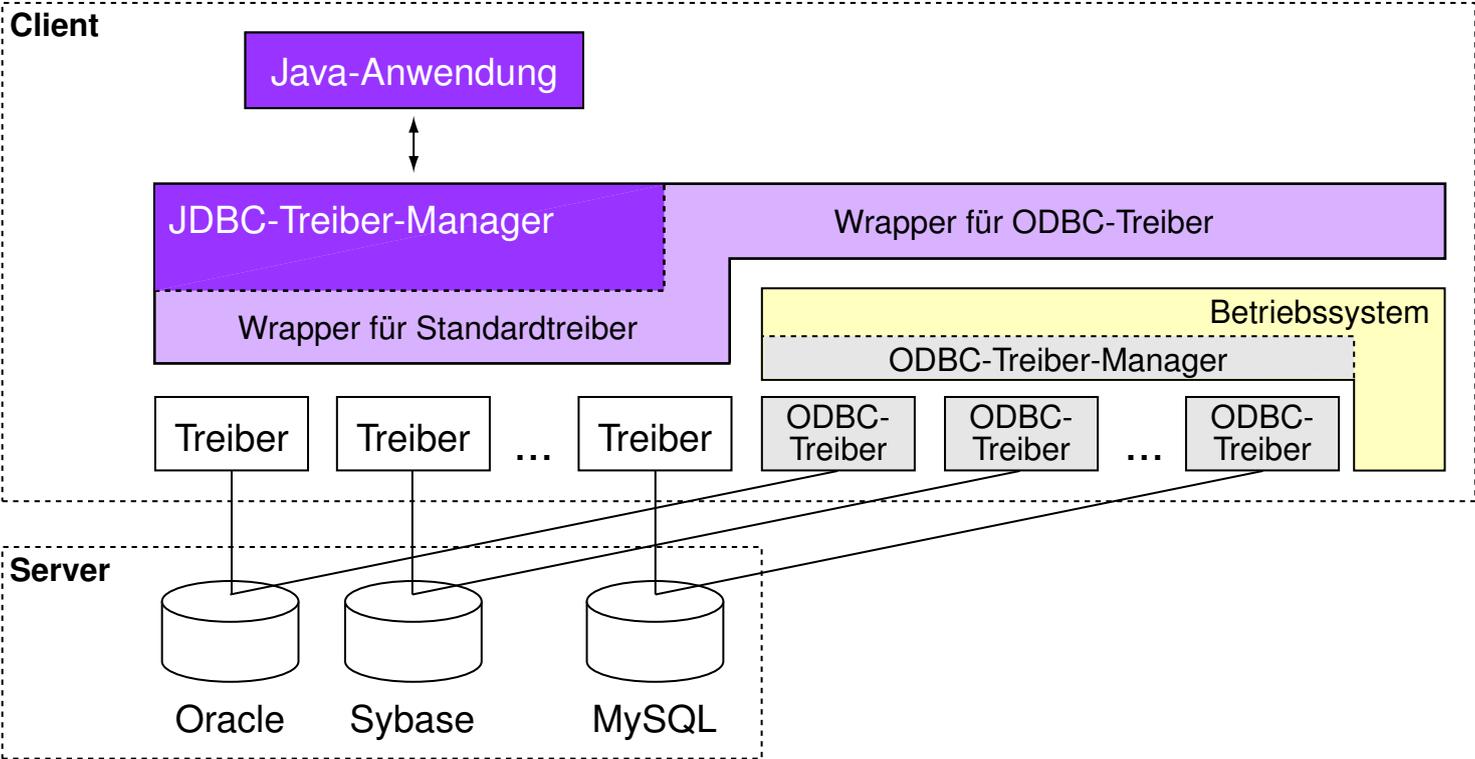
Programmierschnittstelle (Fortsetzung)



In Java sind die Funktionen der ODBC-Schnittstelle als Teil des JDBC-Treiber-Managers implementiert.

# SQL vom Programm aus

Programmierschnittstelle (Fortsetzung)



In Java sind die Funktionen der ODBC-Schnittstelle als Teil des JDBC-Treiber-Managers implementiert.

# SQL vom Programm aus

## Programmierschnittstelle: JDBC

Gegenüberstellung wichtiger Anwendungsoperationen, ODBC-Funktionsnamen und der JDBC-Implementierung:

---

Operation (aus Anwendungssicht)	ODBC-Funktionsname	Implementierung in JDBC Klasse: Methode
Verbindung zu DBMS aufbauen	SQLConnect	DriverManager: getConnection()
SQL-Anfrage ausführen	SQLExecute	Statement: executeQuery()
Ergebnisse abholen	SQLFetch	ResultSet: next()
Fehlermeldung abfragen	SQLError	SQLException
Transaktion deklarieren	SQLTransact	Connection: setAutoCommit()
Transaktion ausführen	SQLTransact	Connection: commit()
Transaktion zurücknehmen	SQLTransact	Connection: rollback()
Verbindung zu DBMS trennen	SQLDisconnect	Connection: close()

---

## Bemerkungen:

- ❑ ODBC is an API specification for using SQL queries to access data. An implementation of ODBC will contain one or more applications, a core ODBC library, and one or more database drivers. The core library is independent of the applications and DBMSes, and acts as an “interpreter” between the applications and the database drivers. The DBMS-specific details are contained in the database drivers. Thus, it is possible to write applications that use standard types and features without concern for the specifics of each DBMS that might be used. Likewise, database driver implementors need only know how to attach to the core library. [ <http://en.wikipedia.org/wiki/Odbc> ]
- ❑ Ein ODBC-Treiber macht eine Datenquelle (z.B. eine MySQL-Datenbank) zu einer *ODBC-Datenquelle*. ODBC-Datenquellen verstehen ODBC-Funktionsaufrufe. ODBC-Treiber gibt es für die gängigen Betriebssysteme und für fast alle Datenbank-Management-Systeme.
- ❑ ODBC kann von zwei Standpunkten aus betrachtet werden: (a) von einer Anwendung, die in der Lage ist, mit einer ODBC-Datenquelle zu kommunizieren; (b) von einer Datenquelle, die ODBC-Anfragen verstehen und bedienen kann.
- ❑ Microsoft-Äquivalente zu JDBC sind OLE DB und ADO.NET; das vergleichbare Programmpaket für die Programmiersprache Perl heißt DBI.
- ❑ Eine generelle Problematik bei Programmierschnittstellen ist die Verarbeitung von Tupelmengen, die als Ergebnis einer Anfrage geliefert werden. Eine Lösung hierzu bietet das *Cursor-Prinzip*, das in Java (SQLJ, JDBC) als Iterator-Objekt realisiert ist.

# SQL vom Programm aus

## Programmierschnittstelle: JDBC

```
package javadb;
import java.sql.*;

public class JDBCdemo {
    public JDBCdemo() throws ClassNotFoundException {
        // Load JDBC-ODBC-bridge driver.
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }

    public void sampleQuery(String url, String user, String pass, String query)
    throws SQLException{
        Connection con=DriverManager.getConnection(url, user, pass);
        Statement stmt = con.createStatement();
        ResultSet rs=stmt.executeQuery(query);
        while (rs.next()) {
            String n = rs.getString("Name");
            int c = rs.getInt("ChefPersNr");
            System.out.println(n + " " + c);
        }
        con.close();
    }
}
```

...

# SQL vom Programm aus

## Programmierschnittstelle: JDBC (Fortsetzung)

```
public static void main(String[] args){
    JDBCDEMO jd=null;
    try{
        jd=new JDBCDEMO();
        String url="jdbc:odbc:mitarbeiterdb"; // mitarbeiterdb is data source name
        String user="stein";
        String pass="";
        String query="select Name, ChefPersNr ";
            query+="from Mitarbeiter ";
            query+="where ChefPersNr < 8000";
        jd.sampleQuery(url, user, pass, query);
    }
    catch(ClassNotFoundException cnfe){
        throw new RuntimeException(cnfe);
    }
    catch(SQLException sqle){
        throw new RuntimeException(sqle);
    }
}
}
```

```
[stein@pc]$ java javadb.JDBCDEMO
Smith 3334
```

# SQL vom Programm aus

## MySQL Version 4.1

### Besonderheiten:

- ❑ `create table ...`  
`( ... ) type=InnoDB;`

### Einschränkungen (u. a.):

- ❑ keine Deklaration von Domains
- ❑ keine Check-Klausel
- ❑ Update-Klausel darf keinen SFW-Block enthalten

### Download:

- ❑ <http://dev.mysql.com/downloads/>

# SQL vom Programm aus

## MySQL Version 5.0

Neuerungen gegenüber Version 4.1 (u. a.) :

- ❑ Stored-Procedures, Stored-Functions
- ❑ Default-Storage-Engine ist InnoDB
- ❑ eingeschränkte Konzepte für Trigger
- ❑ benannte und aktualisierbare Sichten
- ❑ Cursor