Kapitel DB:VI

VI. Die relationale Datenbanksprache SQL

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

Einführung

Historie

1974 Ausgangspunkt ist SEQUEL, die *Structured English Query Language*

⋮ Wildwuchs verschiedener Dialekte

1986 SQL1 wird von der ANSI als Standard verabschiedet

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 Vewendung von SQL in Zusammenhang mit XML

2008 SQL:2008 mit INSTEAD OF-Trigger und TRUNCATE-Statement

2011 SQL:2011 ist die aktuelle Revision

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 Aggregierung, 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: Art des Index, Zugriffsoptimierung

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,
   Sperren 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 [SFW-Block Intro]

Buch		
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 Buch
```

$\rightsquigarrow$

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 [SFW-Block Intro]

Buch		
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 Buch
```

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

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
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 Buch
where Verlag = 'Addison Wesley'
```

$\rightsquigarrow$

Titel	Verlag
Fundamentals of Database Systems	Addison Wesley
Heuristics	Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
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 Buch
```

$\rightsquigarrow$

Verlag
Piper
Berlin-Verlag
Addison Wesley
Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
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 Buch
```

$\rightsquigarrow$

Verlag
Piper
Berlin-Verlag
Addison Wesley

SQL als Datenanfragesprache

Syntax des SFW-Blocks

```
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:

- Die dargestellte Syntax enthält die wichtigsten Elemente. Moderne Datenbanksysteme stellen noch eine Reihe von Erweiterungen zur Verfügung, die über das hier notierte Schema hinausgehen. Beispiel: [MySQL]
- `[[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 [SFW-Block-Syntax]

```
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 auch die mehrfache Spezifikation von demselben Attribut einer 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 as m, Mitarbeiter as employee
```

$\rightsquigarrow$ Bildung des kartesischen Produktes,
Ausgabe einer Tabelle mit 10 Spalten und 16 Zeilen

SQL als Datenanfragesprache

Select-Klausel [SFW-Block-Syntax]

```
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 Aliasen bedeutet eine Umbennung 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 (aufwändige) 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 m
```

```
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
from Mitarbeiter
```

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

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

```
select m.PersNr
from Mitarbeiter as m
```

Unerlaubte Anfrage wegen Mehrdeutigkeit von `PersNr` :

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

SQL als Datenanfragesprache

Where-Klausel [SFW-Block-Syntax]

```
[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 α im Tupelkalkül oder im Domänenkalkül. 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

SQL als Datenanfragesprache

Where-Klausel [SFW-Block-Syntax]

```
[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 α im Tupelkalkül oder im Domänenkalkül. 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
```

$\rightsquigarrow$

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
```

$\rightsquigarrow$

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5

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

$\rightsquigarrow$

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
$\rightsquigarrow$	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

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
$\rightsquigarrow$	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';
```

Variante mit Subquery in From-Klausel

```
select m1.Name
from Mitarbeiter as m1,
    (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 π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

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 π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

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 π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

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:

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

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

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 π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

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 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:

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

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:

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

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

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

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:

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

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

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

- SQL-Where entspricht einem als Formel spezifiziertem Constraint:

$$\{(t_3.A_1, t_2.A_3) \mid r_3(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

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

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

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

Beispiel:

```
select m1.Name
from Mitarbeiter as m1
where exists
      (select * from Mitarbeiter as m2
       where m2.Name = 'Smith' and
             m2.AbtNr = m1.AbtNr and
             m1.Name != 'Smith');
```

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

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

2. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \ldots)\}$ [not] in
   (**select... from... where...**)

3. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \ldots)\}$ {=|<>|<| ...} [any | all]
   (**select... from... where...**)

Bemerkungen:

- Geschachtelte Anfragen heißen auch Unterabfragen oder Subqueries. Subqueries erhöhen nicht die Ausdruckskraft von SQL, sondern erleichtern nur die Formulierung von Anfragen. Die Semantik jeder Subquery lässt sich mit Join-Operationen nachbilden.
- Subqueries in der From-Klausel *(Derived Table Subqueries)* dienen zur Bildung spezialisierter Tabellen für kartesische Produkte. [Beispiel]
- Subqueries in der Where-Klausel *(Expression Subqueries)* dienen zur Formulierung von Bedingungen. Wichtige Verwendungsformen:
  1. Das Ergebnis der Subquery wird daraufhin getestet, ob es die leere Menge ist, d.h., ob es einen oder keinen Match gibt. [Beispiel]
  2. Das Ergebnis der Subquery wird daraufhin getestet, ob es einen bestimmten Attributwert oder ein bestimmtes Tupel enthält.
  3. Ohne `any` bzw. `all`. Das Ergebnis der Subquery muss genau *ein* Element zurückliefern, das dann bzgl. der angegebenen Relation ($=, <>, <, \ldots$) 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`.

Bemerkungen (Fortsetzung):

- Subqueries können weiter geschachtelt werden, 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 zu semantisch äquivalenten Anfrageformulierungen stark unterschiedliche Antwortzeiten.

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 die Kurs- und Angebotsnummern der Teilnehmer aus 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 die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

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

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 die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

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

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 die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

SQL Variante (b), korrelierte Unterabfrage

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where exists    [Subquery-Verwendungsform 1]
       (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

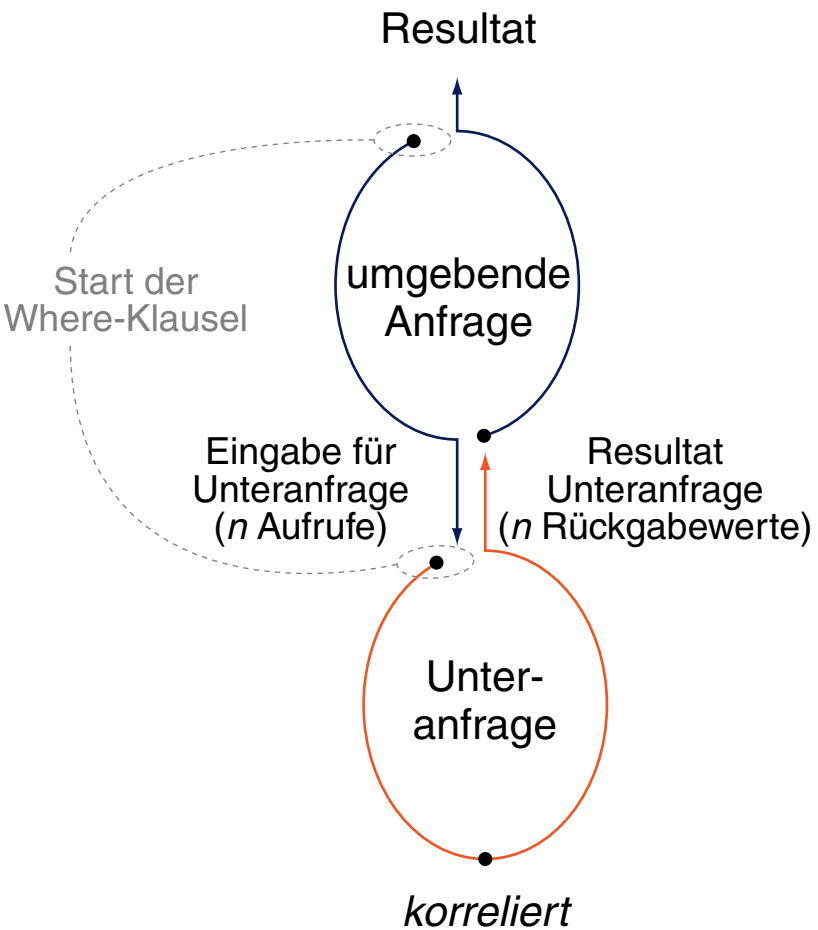
„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

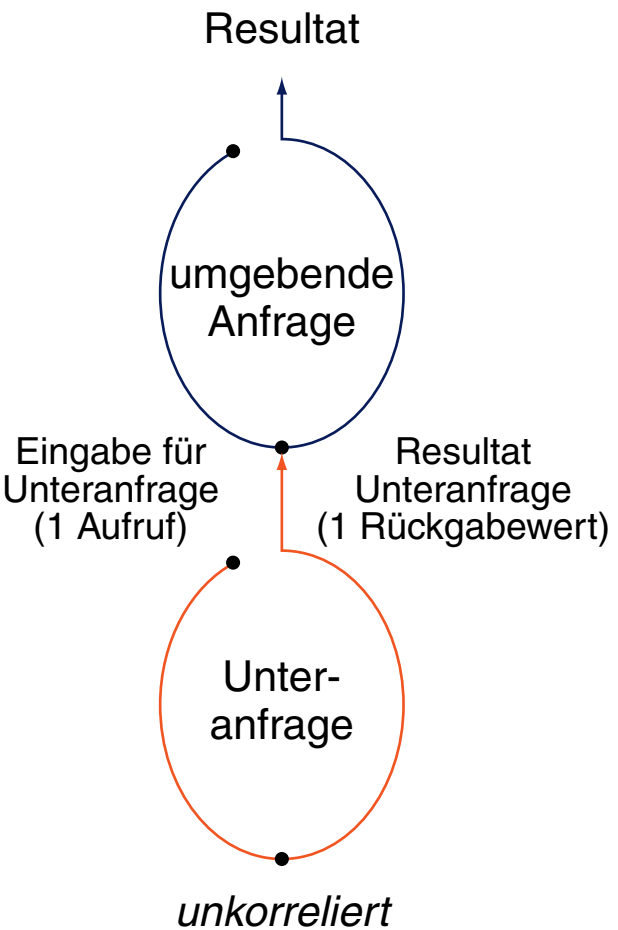
SQL Variante (c), unkorrelierte Unterabfrage

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where nt.TeilnNr in    [Subquery-Verwendungsform 2]
       (select TeilnNr
        from Teilnehmer
        where Ort = 'Bremen')
```

SQL als Datenanfragesprache

Geschachtelte Anfragen





[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		
TeilNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilNr
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

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

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

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

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

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

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

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

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

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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
      (select *
       from Kurs t3
       where not exists
             (select *
              from nimmt_teil t2
              where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
      (select *
       from Kurs t3
       where not exists
             (select *
              from nimmt_teil t2
              where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.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

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

SQL

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (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 Sätzen aus der Sprache selbst und der Formulierung von Zusammenhängen *über* solche Sätze. Sätze aus der Sprache selbst dienen uns zur Kommunikation mittels dieser Sprache; die Symbole, die vewendet werden, um solche Sätze zu formulieren, gehören zur Objektsprache. Symbole, die verwendet werden, um *über* Sätze zu sprechen, die in der Objektsprache formuliert sind, gehören zur Metasprache. [DB:V Formelsemantik]
- Die Formelbezeichner α, β, γ, die Prädikatsbezeichner P, Q, die Quantoren $\forall$, $\exists$, die Variablenbezeichner t, x, y, z, und die Junktoren $\neg$, $\wedge$, $\vee$, $\rightarrow$ gehören zur Objektsprache. Das $\approx$-Zeichen ist ein Zeichen der Metasprache und steht für „ist logisch äquivalent mit“. Es gelten u.a. folgende Zusammenhänge: [DB:V Formelsemantik]

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

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

  $$\neg(\alpha \vee \beta) \approx \neg\alpha \wedge \neg\beta \quad \text{bzw.} \quad \neg(\alpha \wedge \beta) \approx \neg\alpha \vee \neg\beta$$

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

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 A_2 **from** r_1 **where** <condition1>
  union [all]
  select A_2 **from** r_2 **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 A_2 **from** r_1 **where** <condition1>
  union [all]
  select A_2 **from** r_2 **where** <condition2>

- Durchschnitt.

  select A_2 **from** r_1, r_2
  where $r_1.A_2 = r_2.A_2$ 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    [Subquery-Verwendungsform 2]
      (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.
- `union` macht eine Duplikatelimierung; `union all` konstruiert eine Multimenge.

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen

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

Aggregatfunktionen in SQL-89:

Name	Beschreibung
`count(*)`	Anzahl der Tupel
`count([distinct] <attribute>)`	Anzahl der Attributausprägungen
`max(<attribute>)`	Maximum der Attributausprägungen
`min(<attribute>)`	Minimum der Attributausprägungen
`avg([distinct] <attribute>)`	Durchschnitt der Attributausprägungen
`sum([distinct] <attribute>)`	Summe der Attributausprägungen

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
from
where
```

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 =    [Subquery-Verwendungsform 3]
      (select max(Alter)
       from Kursleiter)
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen

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

nimmt_teil		
AngebotsNr	KursNr	TeilNr
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 =    [Subquery-Verwendungsform 3]
      (select max(Alter)
       from Kursleiter)
```

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 <              [Subquery-Verwendungsform 3]
      (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 =    [Subquery-Verwendungsform 3]
       (select max(Alter)
        from Kursleiter)
```

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 < exists    [Subquery-Verwendungsform 1]
       (select count(*) *
        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 Gruppen- bzw. Teilmengenbildung: alle Tupel, die gleiche Werte in der spezifizierten Attributliste haben, bilden eine Teilmenge.
- Mittels der Having-Klausel lassen sich Nebenbedingungen für die Teilmengen formulieren.
- In der Select-Klausel dürfen nur Aggregatfunktionen oder Attribute, nach denen gruppiert wurde, vorkommen.
- Aggregatfunktionen werden auf die Teilmengen angewandt.

Bemerkungen:

- Unterschied zwischen der Where-Klausel und der Having-Klausel: Mit der Where-Klausel werden Tupel gefiltert, mit der Having-Klausel werden Tupel*mengen* gefiltert.
- *Logische* Ausführungsreihenfolge: from $\rightarrow$ where $\rightarrow$ group by $\rightarrow$ having $\rightarrow$ select
- Aggregatfunktionen können nicht geschachtelt werden.

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?“

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
```

$\rightsquigarrow$

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
```

$\rightsquigarrow$

Anzahl	Faehigkeit
2	Java
1	C++
1	Linux

SQL als Datenanfragesprache

Nullwerte

Der spezielle Wert „Null“, $\perp$, ist als Wert in jedem Datentyp vorhanden. Gründe, die zur Verwendung von Null als Attributausprägung führen:

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. Gründe, die zur Verwendung von Null als Attributausprägung führen:

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.

Operationen mit Null:

(a) Eine arithmetische Operation ergibt Null, falls *ein* Operand Null ist.

(b) Für den Test auf Null dienen die Operatoren `is` und `is not`.

	Beispiele	Wert
(a)	`4+null, 4<null, null=null`	`null`
(b)	`null is null`	`true`
(b)	`null is not null`	`false`

SQL als Datenanfragesprache

Nullwerte

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

Boolsche Semantik:

¬	
true	false
unknown	unknown
false	true

SQL-Entsprechung:

not	
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	$\rightsquigarrow$ TAFEL		
unknown	$\rightsquigarrow$ TAFEL		
false	$\rightsquigarrow$ TAFEL		

SQL-Entsprechung:

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	$\rightsquigarrow$ TAFEL		
null	$\rightsquigarrow$ TAFEL		
0	$\rightsquigarrow$ TAFEL		

Bemerkungen:

- In arithmetischen Ausdrücken werden Nullwerte propagiert.
- 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 einer eigenen Gruppe zusammengefasst.

SQL als Datenanfragesprache

Zusammengesetzte Terme

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

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

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

$\rightsquigarrow$

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
```

$\rightsquigarrow$

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)

VI. Die relationale Datenbanksprache SQL

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

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 = Anzahl aller Stellen, 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 wird. Dient zur Formulierung von Zeitintervallen.

SQL als Datendefinitionssprache

Domains (SQL-92)

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

- `<datatype>` bezeichnet einen integrierten Datentyp einschließlich eventueller Stellenanzahl.
- 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 einschließlich eventueller Stellenanzahl.
- 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 für Domains, Attribute und Relationenschemata.
- Die Check-Klausel erlaubt die Formulierung von Prädikaten, wie sie in der Where-Klausel bei Anfrageoperationen spezifiziert werden können.

Beispiele: [Relationen definieren]

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

check(Alter between 18 and 27)

check(Dauer > 3)
```

SQL als Datendefinitionssprache

Check-Klausel

```
check (<condition>)
```

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

Beispiele: [Relationen definieren]

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

check(Alter between 18 and 27)

check(Dauer > 3)

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 Performanz 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>]
)
```

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>)] |
                     check (<condition>)}
```

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>)] |
                     check (<condition>)}
```

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

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>)] |
                     check (<condition>)}
```

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

```
<action> ::= [on update <type>] [on delete <type>]
```

Bemerkungen:

- Mittels `default` lässt sich ein Standardwert für Attribute vorgeben.
- Integritätsbedingungen beziehen sich auf ein Tupel, eine Relation oder mehrere Relationen. Besteht ein Schlüssel aus einem einzigen Attribut, so kann die entsprechende Integritätsbedingung mit `<attr_int_cond>` direkt innerhalb der Attributdeklaration vereinbart werden.
- Integritätsbedingungen für Schlüssel, die aus mehreren Attributen bestehen, werden mittels `<rel_int_cond>` vereinbart.
- In SQL-92 sind Primary-Key-Attribute implizit „unique“ und „not null“.

SQL als Datendefinitionssprache

Relationen definieren: Beispiele

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

SQL als Datendefinitionssprache

Relationen definieren: Beispiele

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

Verwendung der Form `<attr_int_cond>`:

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

Verwendung der Form `<rel_int_cond>`:

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

SQL als Datendefinitionssprache

Relationen definieren: Beispiele [Check-Klausel]

```
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 Buch (ISBN),


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

)
```

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

```
<action> ::= [on update <type>] [on delete <type>]
```

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
- `set null`
- `set default`
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

`<action&gt` ::= [on update <type>] [on delete <type>]

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.
- `set null`
- `set default`
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

`<action&gt` ::= [on update <type>] [on delete <type>]

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.
- `set null`
  Setzt die Fremdschlüsselwerte auf Null, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `set default`
  Setzt die Fremdschlüsselwerte auf den Wert, der in der Default-Klausel des Fremdschlüsselattributes angegeben ist, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

`<action&gt` `::= [on update <type>] [on delete <type>]`

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.
- `set null`
  Setzt die Fremdschlüsselwerte auf Null, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `set default`
  Setzt die Fremdschlüsselwerte auf den Wert, der in der Default-Klausel des Fremdschlüsselattributes angegeben ist, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `restrict` (Default)
  Erzeugt eine Fehlermeldung bei dem Versuch, einen Primärschlüsselwert zu ändern oder zu löschen, wenn dieser als Fremdschlüsselwert referenziert wird.

SQL als Datendefinitionssprache

Relationen ändern

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

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

SQL als Datendefinitionssprache

Relationen ändern

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

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

```
<add_cons> ::= {add constraint unique (<attribute1>, ...) |
               add constraint primary key (<attribute1>, ...) |
               add constraint foreign key (<attribute1>, ...)
                                 references <table> (<attribute1>, ...) |
               add constraint check (<condition>)}
```

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   |
 [(<attribute1>, ...)] values <SFW-Block>
}
```

- Einfügen von vollständigen oder unvollständigen Tupeln. Die Tupel können explizit oder mittels eines Select-From-Where-Blocks spezifiziert werden.
- Alle einzufügenden Werte müssen die Integritätsbedingungen erfüllen.

SQL als Datenmanipulationsssprache

Einfügen von Tupeln

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

- Einfügen von vollständigen oder unvollständigen Tupeln. Die Tupel können explizit oder mittels eines Select-From-Where-Blocks spezifiziert 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 den Attributnamen. Fehlende Attribute erhalten den Nullwert.

SQL als Datenmanipulationsssprache

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

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

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

```
insert into Kursgebuehr values (1, G08, 200, null)
```
bzw.
```
insert into Kursgebuehr (AngNr, KursNr, TnNr) values (1, G08, 200)
```

SQL als Datenmanipulationsssprache

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

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

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

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

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

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

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

```
insert into Kursgebuehr values (1, G08, 200, null)   bzw.
insert into Kursgebuehr (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 Kursgebuehr 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 Kursgebuehr (AngNr, KursNr, TnNr)
select *
from nimmt_teil
```

SQL als Datenmanipulationsssprache

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 Datenmanipulationsssprache

Löschen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Lösche alle Tupel aus der nimmt_teil-Relation.“

```
delete
from nimmt_teil
```

SQL als Datenmanipulationssprache

Löschen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Lösche alle Tupel aus der nimmt_teil-Relation.“

```
delete
from nimmt_teil
```

„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 Datenmanipulationsssprache

Ändern von Tupeln

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

SQL als Datenmanipulationssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

SQL als Datenmanipulationsssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

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

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

SQL als Datenmanipulationssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

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

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

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

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

SQL als Datenmanipulationsssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

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

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

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

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

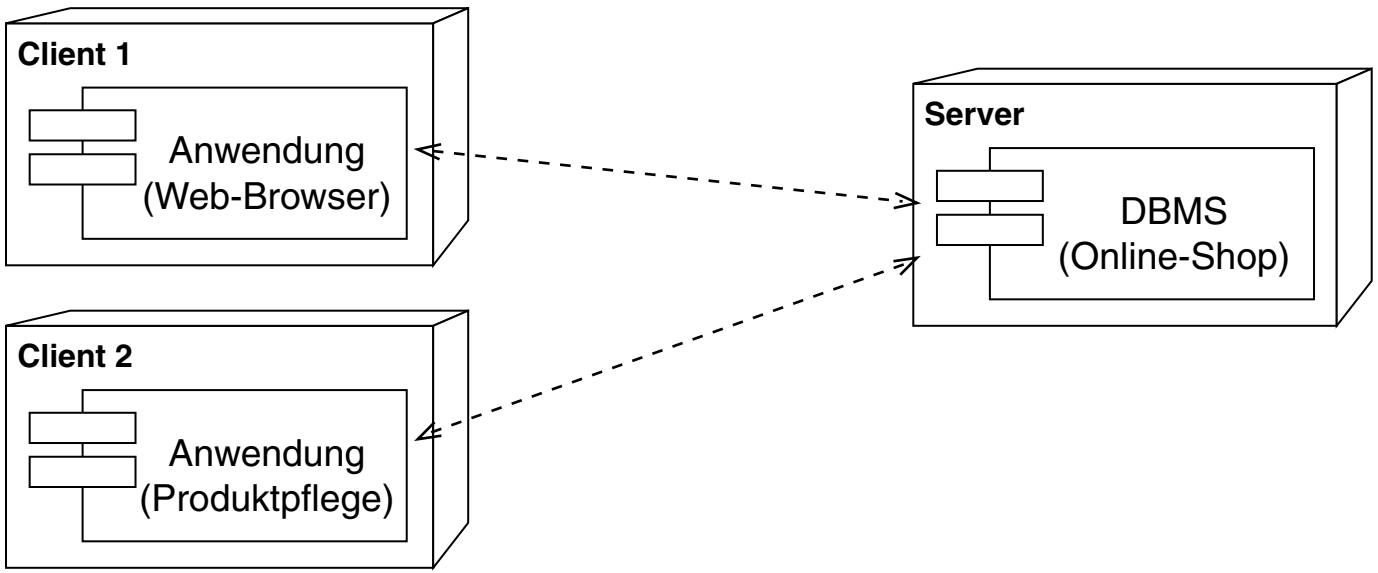
Bemerkungen:

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

VI. Die relationale Datenbanksprache SQL

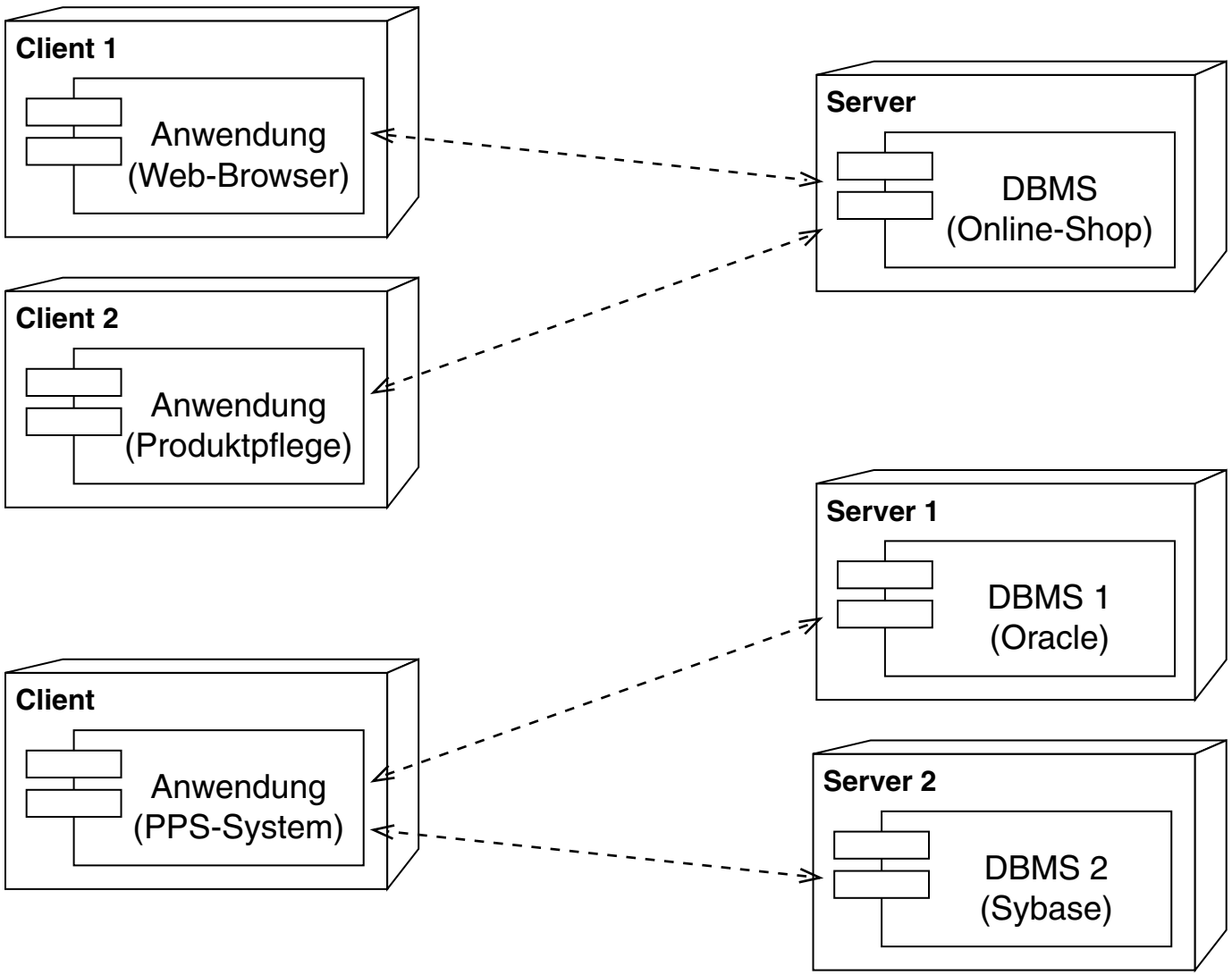
SQL vom Programm aus

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
   - Prinzip: SQL-Anweisungen werden im Programmquelltext ausgezeichnet
   - Vorteil: (SQL-)Statements lassen sich zur Übersetzungszeit überprüfen und optimieren
   - Beispiel: Embedded SQL; Realisierung für die Programmiersprache Java heißt SQLJ
   - Erweiterung: Dynamic SQL

SQL vom Programm aus

Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen

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

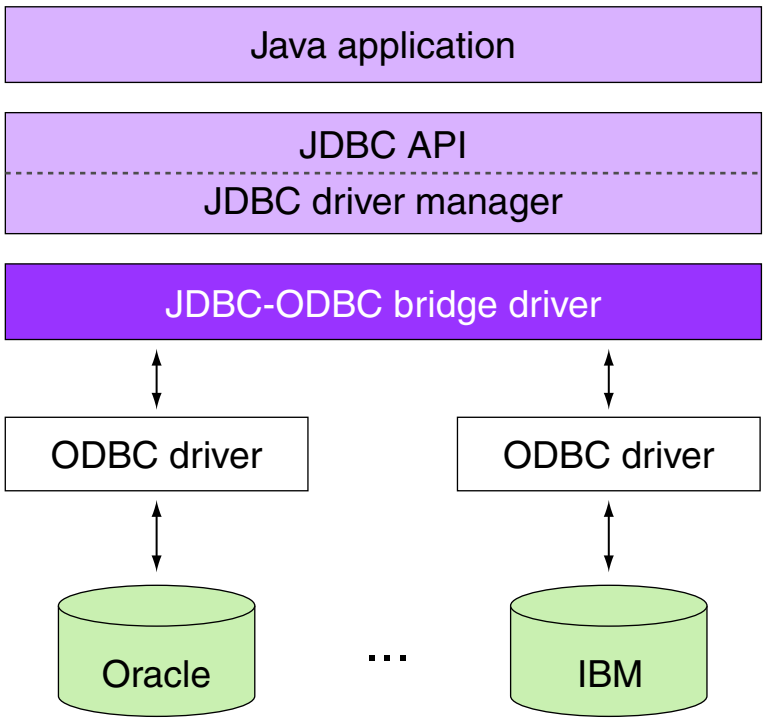
3. Programmierschnittstelle / API *(Application Programming Interface)*
   - Prinzip: SQL-Anweisungen werden als zur Laufzeit generierbarer Text an das Datenbanksystem übergeben
   - Vorteil: hohe Flexibilität

Bemerkungen:

- 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.
- Java Database Connectivity, JDBC, ist eine Datenbankschnittstelle der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist. [wikipedia]

SQL vom Programm aus

JDBC-Programmierschnittstelle: Treibertypen



- JDBC-Typ-1-Treiber. [Typ 2] Übersetzung von JDBS-Aufrufen in ODBC-Aufrufe mittels eines sogenannten JDBC-ODBC-Bridge-Treibers.

  JDBC = Java Database Connectivity
  ODBC = Open Database Connectivity

SQL vom Programm aus

JDBC-Programmierschnittstelle: Vergleich mit ODBC

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

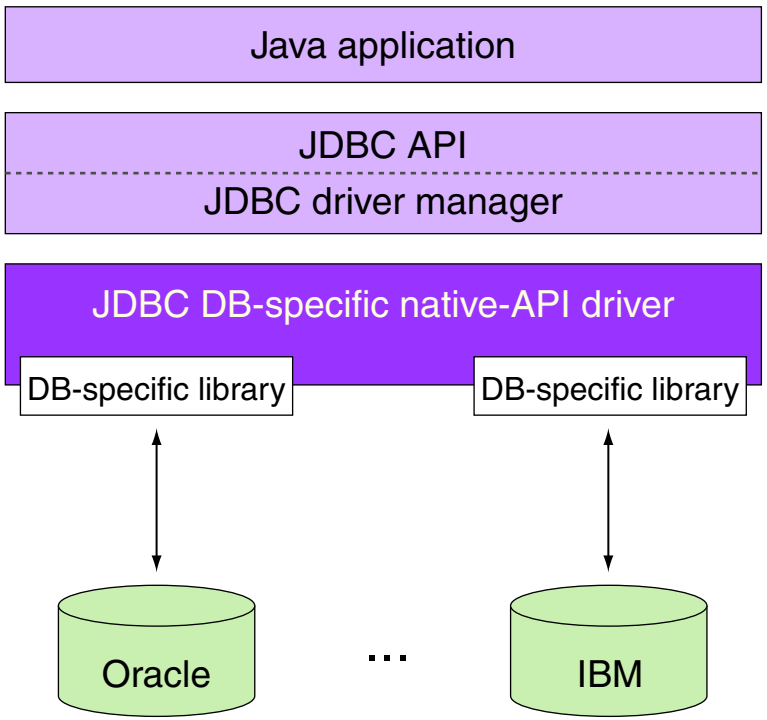
Operation (Anwendungssicht)	ODBC-Funktionsname	Implementierung in JDBC <class>:<method>
Verbindung zu DBMS aufbauen	SQLConnect	DriverManager: getConnection()
SQL-Anfrage ausführen	SQLExecute	Statement: executeQuery()
Ergebnisse abholen	SQLFetch	ResultSet: next()
Fehlermeldung abfragen	SQLError	SQLException
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 a standard programming language middleware API for DBMS. ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An application that can use ODBC is referred to as “ODBC-compliant”. Any ODBC-compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMSs and even for text or CSV files. [wikipedia]
- Ein ODBC-Treiber macht eine Datenquelle (z.B. eine MySQL-Datenbank oder eine einfache Datei) zu einer ODBC-Datenquelle, die ODBC-Funktionsaufrufe versteht. ODBC lässt sich von zwei Standpunkten aus betrachten: (a) von einer Anwendung aus, die in der Lage ist, mit einer ODBC-Datenquelle zu kommunizieren, (b) von einer Datenquelle aus, die ODBC-Anfragen verstehen und bedienen kann.

SQL vom Programm aus

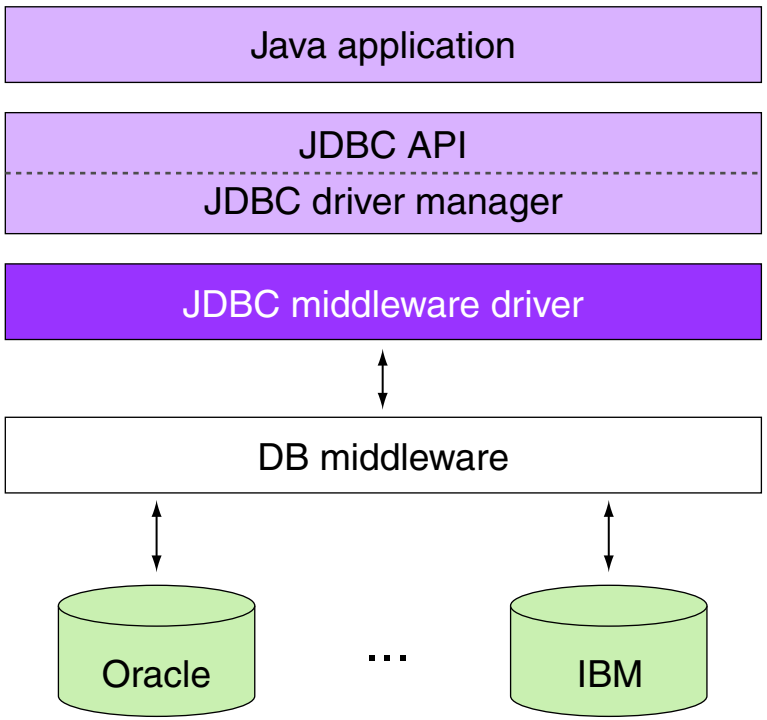
JDBC-Programmierschnittstelle: Treibertypen (Fortsetzung)



- JDBC-Typ-2-Treiber. [Typ 1] Übersetzung von JDBS-Aufrufen in Aufrufe für einen Datenbankserver mittels einer plattform- und datenbankspezifischen Programmbibliothek.

SQL vom Programm aus

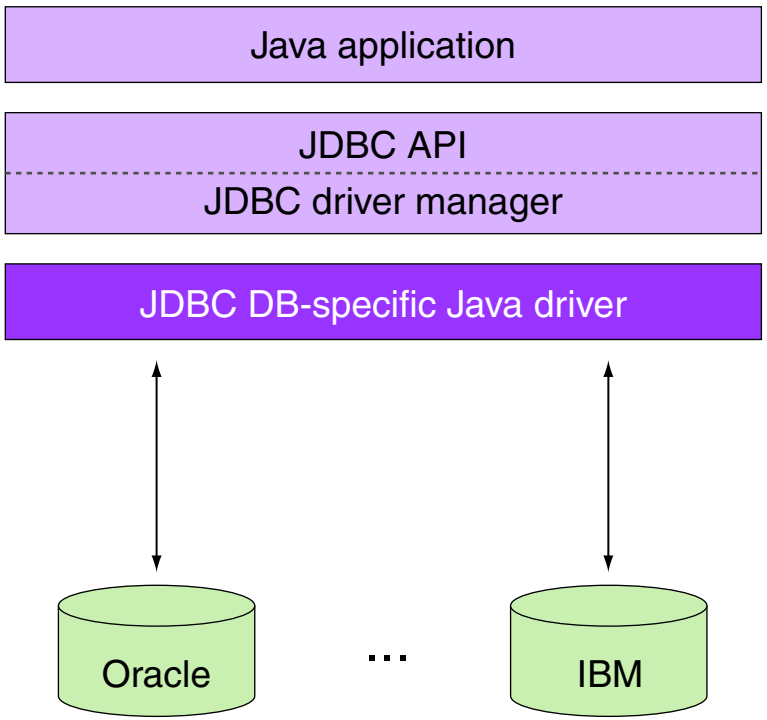
JDBC-Programmierschnittstelle: Treibertypen (Fortsetzung)



- JDBC-Typ-3-Treiber. Übersetzung von JDBC-Aufrufen in generische DBMS-Aufrufe sowie Übermittlung an die Middleware eines Anwendungsservers, welche die Aufrufe dann für spezifische Datenbankserver übersetzt.

SQL vom Programm aus

JDBC-Programmierschnittstelle: Treibertypen (Fortsetzung)



- JDBC-Typ-4-Treiber. Übersetzung von JDBC-Aufrufen für einen spezifischen Datenbankserver ohne Verwendung einer plattform- und datenbankspezifischen Programmbibliothek.

SQL vom Programm aus

JDBC-Programmierschnittstelle: Beispiel (Typ-4-Treiber)

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

public class JdbcDemo {

 public JdbcDemo() throws ClassNotFoundException {
  // Requires MySQL Connector/J.
  Class.forName("com.mysql.jdbc.Driver");
 }

 public ResultSet submitQuery(
         String url, String user, String pass, String query)
         throws SQLException {
  Connection connection = DriverManager.getConnection(url, user, pass);
  Statement statement = connection.createStatement();
  ResultSet result = statement.executeQuery(query);
  return result;
 }
```

SQL vom Programm aus

JDBC-Programmierschnittstelle: Beispiel (Fortsetzung)

```
 public static void main(String[] args) throws Exception {
   JdbcDemo demo = new JdbcDemo();
   String db     = "mitarbeiterdb";
   String url    = "jdbc:mysql://pcstein.medien.uni-weimar.de/" + db;
   String user   = "stein";
   String pass   = "";
   String query  = "select Name, ChefPersNr "
                 + "from mitarbeiter "
                 + "where ChefPersNr < 8000";

   ResultSet result = demo.submitQuery(url, user, pass, query);

   while (result.next()) {
    String name = result.getString("Name");
    int chefPersNr = result.getInt("ChefPersNr");
    System.out.println(name + ' ' + chefPersNr);
  }
}
```

```
[stein@pcstein]$ javac jdbc/JdbcDemo.java
[stein@pcstein]$ java -cp .:mysql-connector-java.jar jdbc.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:

- dev.mysql.com/downloads/

SQL vom Programm aus

MySQL Version 5.x

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