

Teil VII

Die relationale Anfragesprache SQL

Die relationale Anfragesprache SQL



1. Aufbau von SQL-Anfragen
2. Erweiterungen des SFW-Blocks
3. Aggregatfunktionen und Gruppierungen

Lernziele für heute . . .

- Erweiterte Kenntnisse zum relationalen SQL
- Kenntnisse von Erweiterungen des SFW-Blocks



Aufbau von SQL-Anfragen

Struktur einer SQL-Anfrage

```
-- Anfrage  
select projektionsliste  
from relationenliste  
[ where bedingung ]
```

select

- Projektionsliste
- arithmetische Operationen und Aggregatfunktionen

from

- zu verwendende Relationen, evtl. Umbenennungen

where

- Selektions-, Verbundbedingungen
- Geschachtelte Anfragen (wieder ein SFW-Block)

Auswahl von Tabellen: Die from-Klausel

- einfachste Form; hinter jedem Relationennamen kann optional eine Tupelvariable stehen

```
select *  
from relationenliste
```

- Beispielanfrage:

```
select *  
from WEINE
```

Die select-Klausel

- Festlegung der **Projektionsattribute**

```
select [distinct] projektionsliste  
from ...
```

- mit

```
projektionsliste := { attribut |  
                     arithmetischer-ausdruck |  
                     aggregat-funktion } [, ...]
```

Die select-Klausel: Projektionsliste



- Attribute der hinter `from` stehenden Relationen, optional mit Präfix, der Relationennamen oder Namen der Tupelvariablen angibt
- arithmetische Ausdrücke über Attributen dieser Relationen und passenden Konstanten
- Aggregatfunktionen über Attributen dieser Relationen

Die select-Klausel

- Spezialfall der Projektionsliste: *
 - liefert alle Attribute der Relation(en) aus dem from-Teil

```
select *  
from WEINE
```

distinct eliminiert Duplikate

```
select Name from WEINE
```

- liefert die Ergebnisrelation als Multimenge:

Name
La Rose Grand Cru
Creek Shiraz
Zinfandel
Pinot Noir
Pinot Noir
Riesling Reserve
Chardonnay

distinct eliminiert Duplikate /2

```
select distinct Name from WEINE
```

- ergibt Projektion aus der Relationenalgebra:

Name
La Rose Grand Cru
Creek Shiraz
Zinfandel
Pinot Noir
Riesling Reserve
Chardonnay

Tupelvariablen und Relationennamen

- Anfrage

```
select Name from WEINE
```

- ist äquivalent zu

```
select WEINE.Name from WEINE
```

- und

```
select W.Name from WEINE W
```

Kartesisches Produkt

- bei mehr als einer Relation wird das kartesische Produkt gebildet:

```
select *  
from WEINE, ERZEUGER
```

- **alle Kombinationen werden ausgegeben!**

Tupelvariablen für mehrfachen Zugriff

- Einführung von Tupelvariablen erlaubt mehrfachen Zugriff auf eine Relation:

```
select *  
from WEINE w1, WEINE w2
```

- Spalten lauten dann:

```
w1.WeinID, w1.Name, w1.Farbe, w1.Jahrgang,  
    w1.Weingut,  
w2.WeinID, w2.Name, w2.Farbe, w2.Jahrgang,  
    w2.Weingut
```

- frühe SQL-Versionen
 - üblicherweise realisierter Standard in aktuellen Systemen
 - kennen nur Kreuzprodukt, keinen expliziten Verbundoperator
 - Verbund durch Prädikat hinter where realisieren
- Beispiel für natürlichen Verbund:

```
select *  
from WEINE, ERZEUGER  
where WEINE.Weingut = ERZEUGER.Weingut
```

Verbund explizit: natural join

- neuere SQL-Versionen
 - kennen mehrere explizite Verbundoperatoren (engl. *join*)
 - als Abkürzung für die ausführliche Anfrage mit Kreuzprodukt aufzufassen

```
select *  
from WEINE natural join ERZEUGER
```


Verbunde als explizite Operatoren: join

- Verbund mit beliebigem Prädikat:

```
select *  
from WEINE join ERZEUGER  
      on WEINE.Weingut = ERZEUGER.Weingut
```

- Gleichverbund mit using:

```
select *  
from WEINE join ERZEUGER  
      using (Weingut)
```

Verbund explizit: cross join

- Kreuzprodukt

```
select *  
from WEINE, ERZEUGER
```

- als cross join

```
select *  
from WEINE cross join ERZEUGER
```

Tupelvariable für Zwischenergebnisse

- „Zwischenrelationen“ aus SQL-Operationen oder einem SFW-Block können über Tupelvariablen mit Namen versehen werden

```
select Ergebnis.Weingut  
from (WEINE natural join ERZEUGER) as Ergebnis
```

- für from sind Tupelvariablen Pflicht
- as ist optional

Präfixe für Eindeutigkeit

```
select Name, Jahrgang, Weingut      -- (falsch!)  
from WEINE natural join ERZEUGER
```

- Attribut Weingut existiert sowohl in der Tabelle WEINE als auch in ERZEUGER!
- richtig mit Präfix:

```
select Name, Jahrgang, ERZEUGER.Weingut  
from WEINE natural join ERZEUGER
```

Tupelvariablen für Eindeutigkeit

- bei der Verwendung von Tupelvariablen, kann der Name einer Tupelvariablen zur Qualifizierung eines Attributs benutzt werden:

```
select w1.Name, w2.Weingut  
from WEINE w1, WEINE w2
```

Die where-Klausel

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

- Formen der Bedingung:

- Vergleich eines Attributs mit einer Konstanten:

attribut θ *konstante*

mögliche Vergleichssymbole θ abhängig vom Wertebereich; etwa =, <>, >, <, >= sowie <=.

- Vergleich zwischen zwei Attributen mit kompatiblen Wertebereichen:

attribut1 θ *attribut2*

- logische *Konnektoren* or, and und not

Verbundbedingung

- *Verbundbedingung* hat die Form:

relation1.attribut = relation2.attribut

- Beispiel:

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

- *Bereichsselektion*

attrib between *konstante₁* and *konstante₂*

ist Abkürzung für

attrib \geq *konstante₁* and
attrib \leq *konstante₂*

- schränkt damit Attributwerte auf das abgeschlossene Intervall [*konstante₁*, *konstante₂*] ein
- Beispiel:

```
select * from WEINE  
where Jahrgang between 2000 and 2005
```


- Notation

attribut like *spezialkonstante*

- Mustererkennung in Strings (Suche nach mehreren Teilzeichenketten)
- Spezialkonstante kann die Sondersymbole ‘%’ und ‘_’ beinhalten
 - ‘%’ steht für kein oder beliebig viele Zeichen
 - ‘_’ steht für genau ein Zeichen

Ungewissheitsselektion /2

```
select * from WEINE  
where Name like 'La Rose%'
```

ist Abkürzung für

```
select * from WEINE  
where Name = 'La Rose'  
       or Name = 'La RoseA' or Name = 'La RoseAA' ...  
       or Name = 'La RoseB' or Name = 'La RoseBB' ...  
       ...  
       or Name = 'La Rose Grand Cru' ...  
       or Name = 'La Rose Grand Cru Classe' ...  
       ...  
       or Name = 'La RoseZZZZZZZZZZZZZZZZZZ' ...
```

- Mengenoperationen erfordern kompatible Wertebereiche für Paare korrespondierender Attribute:
 - beide Wertebereiche sind gleich oder
 - beide sind auf character basierende Wertebereiche (unabhängig von der Länge der Strings) oder
 - beide sind numerische Wertebereiche (unabhängig von dem genauen Typ) wie integer oder float
- Ergebnisschema := Schema der „linken“ Relation

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

Mengenoperationen in SQL

- *Vereinigung, Durchschnitt und Differenz* als *union, intersect* und *except*
- orthogonal einsetzbar:

```
select *  
from (select Weingut from ERZEUGER  
      except select Weingut from WEINE)
```

äquivalent zu

```
select *  
from ERZEUGER except corresponding WEINE
```

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

```
select *  
from ERZEUGER except corresponding by (Weingut)  
WEINE
```

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

Mengenoperationen in SQL /2

R

A	B	C
1	2	3
2	3	4

S

A	C	D
2	3	4
2	4	5

R union S

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

R union all S

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

Mengenoperationen in SQL /3

R

A	B	C
1	2	3
2	3	4

S

A	C	D
2	3	4
2	4	5

R union corresponding S

A	C
1	3
2	4
2	3

R union corresponding by (A) S

A
1
2

- für Vergleiche mit Wertemengen notwendig:
 - Standardvergleiche in Verbindung mit den Quantoren `all` (\forall) oder `any` (\exists)
 - spezielle Prädikate für den Zugriff auf Mengen, `in` und `exists`

in-Prädikat und geschachtelte Anfragen

- Notation:

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

- Beispiel:

```
select Name  
from WEINE  
where Weingut in (  
    select Weingut from ERZEUGER  
    where Region = 'Bordeaux')
```

Auswertung von geschachtelten Anfragen

1. Auswertung der inneren Anfrage zu den Weingütern aus Bordeaux
2. Einsetzen des Ergebnisses als Menge von Konstanten in die äußere Anfrage hinter in
3. Auswertung der modifizierten Anfrage

```
select Name from WEINE
where Weingut in (
    'Château La Rose', 'Château La Pointe')
```

Name
La Rose Grand Cru

Auswertung von geschachtelten Anfragen /2



- interne Auswertung: Umformung in einen Verbund

```
select Name  
from WEINE natural join ERZEUGER  
where Region = 'Bordeaux'
```

Negation des in-Prädikats

- Simulation des Differenzoperators

$$\pi_{\text{Weingut}}(\text{ERZEUGER}) - \pi_{\text{Weingut}}(\text{WEINE})$$

durch SQL-Anfrage

```
select Weingut
from ERZEUGER
where Weingut not in (
    select Weingut from WEINE )
```

Mächtigkeit des SQL-Kerns

Relationenalgebra	SQL
Projektion	<code>select distinct</code>
Selektion	<code>where</code> ohne Schachtelung
Verbund	<code>from, where</code> <code>from</code> mit <code>join</code> oder <code>natural join</code>
Umbenennung	<code>from</code> mit Tupelvariable; <code>as</code>
Differenz	<code>where</code> mit Schachtelung <code>except corresponding</code>
Durchschnitt	<code>where</code> mit Schachtelung <code>intersect corresponding</code>
Vereinigung	<code>union corresponding</code>

Erweiterungen des SFW-Blocks

- Erweiterungen des SQL-Blocks
 - innerhalb der `from`-Klausel weitere Verbundoperationen (äußerer Verbund),
 - innerhalb der `where`-Klausel weitere Arten von Bedingungen und Bedingungen mit Quantoren,
 - innerhalb der `select`-Klausel die Anwendung von skalaren Operationen und Aggregatfunktionen,
 - zusätzliche Klauseln `group by` und `having`
- rekursive Anfragen

- Umbenennung von Spalten: *ausdruck* as *neuer-name*
- skalare Operationen auf
 - numerischen Wertebereichen: etwa $+$, $-$, $*$ und $/$,
 - Strings: Operationen wie `char_length` (aktuelle Länge eines Strings), die Konkatination `||` und die Operation `substring` (Suchen einer Teilzeichenkette an bestimmten Positionen des Strings),
 - Datumstypen und Zeitintervallen: Operationen wie `current_date` (aktuelles Datum), `current_time` (aktuelle Zeit), $+$, $-$ und $*$
- bedingte Ausdrücke
- Typkonvertierung

Skalare Ausdrücke: Hinweise



- skalare Ausdrücke können mehrere Attribute umfassen
- Anwendung ist tupelweise: pro Eingabetupel entsteht ein Ergebnistupel

Skalare Ausdrücke: Beispiele

- Ausgabe der Namen aller Grand Cru-Weine

```
select substring(Name from 1 for  
    (char_length(Name) -  
    position('Grand Cru' in Name)))  
from WEINE where Name like '%Grand Cru'
```

Skalare Ausdrücke: Beispiele /2

- Annahme: zusätzliches Attribut HerstDatum in WEINE

```
alter table WEINE add column HerstDatum date
```

```
update WEINE set HerstDatum = date '2004-08-13'  
where Name = 'Zinfandel'
```

- Anfrage:

```
select Name,  
       year(current_date - HerstDatum) as Alter  
from WEINE
```

- case-Anweisung: Ausgabe eines Wertes in Abhängigkeit von der Auswertung eines Prädikats

```
case
  when prädikat1 then ausdruck1
  ...
  when prädikatn-1 then ausdruckn-1
  [ else ausdruckn ]
end
```

Bedingte Ausdrücke: Beispiele

- Einsatz in select- und where-Klausel

```
select case
  when Farbe = 'Rot' then 'Rotwein'
  when Farbe = 'Weiß' then 'Weißwein'
  else 'Sonstiges'
end as Weinart, Name from WEINE
```

- explizite Konvertierung des Typs von Ausdrücken

```
cast(ausdruck as typename)
```

- Beispiel: int-Werte als Zeichenkette für Konkatenationsoperator

```
select cast(Jahrgang as varchar) || 'er ' ||  
       Name as Bezeichnung  
from WEINE
```

- Quantoren: all, any, some und exists
- Notation

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

- all: where-Bedingung wird erfüllt, wenn für **alle** Tupel des inneren SFW-Blocks der θ -Vergleich mit *attribut* true wird
- any bzw. some: where-Bedingung wird erfüllt, wenn der θ -Vergleich mit mindestens einem Tupel des inneren SFW-Blocks true wird

Bedingungen mit Quantoren: Beispiele

- Bestimmung des ältesten Weines

```
select *  
from WEINE  
where Jahrgang <= all (  
    select Jahrgang from WEINE)
```

- alle Weingüter, die Rotweine produzieren

```
select *  
from ERZEUGER  
where Weingut = any (  
    select Weingut from WEINE  
    where Farbe = 'Rot')
```


Vergleich von Wertemengen

- Test auf Gleichheit zweier Mengen allein mit Quantoren nicht möglich
- Beispiel: „Gib alle Erzeuger aus, die sowohl Rot- als auch Weißweine produzieren.“
- falsche Anfrage

```
select Weingut  
from WEINE  
where Farbe = 'Rot' and Farbe = 'Weiß'
```

Vergleich von Wertemengen /2

- richtige Formulierung

```
select w1.Weingut
from WEINE w1, WEINE w2
where w1.Weingut = w2.Weingut
      and w1.Farbe = 'Rot' and w2.Farbe = 'Weiß'
```

Das exists/not exists-Prädikat



- einfache Form der Schachtelung

exists (*SFW-block*)

- liefert true, wenn das Ergebnis der inneren Anfrage **nicht** leer ist
- speziell bei **verzahnt geschachtelten** (korrelierte) Anfragen sinnvoll
 - in der inneren Anfrage wird Relationen- oder Tupelvariablen-Name aus dem from-Teil der äußeren Anfrage verwendet

Verzahnt geschachtelte Anfragen

- Weingüter mit 1999er Rotwein

```
select * from ERZEUGER
where 1999 in (
    select Jahrgang from WEINE
    where Farbe = 'Rot' and
        WEINE.Weingut = ERZEUGER.Weingut)
```

Verzahnt geschachtelte Anfragen: konzeptionelle Auswertung

1. Untersuchung des ersten ERZEUGER-Tupels in der äußeren Anfrage (Creek) und Einsetzen in innere Anfrage
2. Auswertung der inneren Anfrage

```
select Jahrgang from WEINE  
where Farbe='Rot' and WEINE.Weingut = 'Creek'
```

3. Weiter bei 1. mit zweitem Tupel ...

Alternative: Umformulierung in Verbund

Beispiel für exists

- Weingüter aus Bordeaux ohne gespeicherte Weine

```
select * from ERZEUGER e
where Region = 'Bordeaux' and not exists (
    select * from WEINE
    where Weingut = e.Weingut)
```

Aggregatfunktionen und Gruppierungen

Aggregatfunktionen und Gruppierung



- Aggregatfunktionen berechnen neue Werte für eine gesamte Spalte, etwa die Summe oder den Durchschnitt der Werte einer Spalte
- Beispiel: Ermittlung des Durchschnittspreises aller Artikel oder des Gesamtumsatzes über alle verkauften Produkte
- bei zusätzlicher Anwendung von Gruppierung: Berechnung der Funktionen pro Gruppe, z.B. der Durchschnittspreis pro Warengruppe oder der Gesamtumsatz pro Kunde

- Aggregatfunktionen in Standard-SQL:
 - `count`: berechnet Anzahl der Werte einer Spalte oder alternativ (im Spezialfall `count(*)`) die Anzahl der Tupel einer Relation
 - `sum`: berechnet die Summe der Werte einer Spalte (nur bei numerischen Wertebereichen)
 - `avg`: berechnet den arithmetischen Mittelwert der Werte einer Spalte (nur bei numerischen Wertebereichen)
 - `max` bzw. `min`: berechnen den größten bzw. kleinsten Wert einer Spalte

- Argumente einer Aggregatfunktion:
 - ein Attribut der durch die `from`-Klausel spezifizierten Relation,
 - ein gültiger skalarer Ausdruck oder
 - im Falle der `count`-Funktion auch das Symbol `*`

- vor dem Argument (außer im Fall von `count(*)`) optional auch die Schlüsselwörter `distinct` oder `all`
 - `distinct`: vor Anwendung der Aggregatfunktion werden doppelte Werte aus der Menge von Werten, auf die die Funktion angewendet wird
 - `all`: Duplikate gehen mit in die Berechnung ein (Default-Voreinstellung)
 - Nullwerte werden in jedem Fall vor Anwendung der Funktion aus der Wertemenge eliminiert (außer im Fall von `count(*)`)

Aggregatfunktionen - Beispiele

- Anzahl der Weine:

```
select count(*) as Anzahl  
from WEINE
```

ergibt

Anzahl
7

Aggregatfunktionen - Beispiele /2

- Anzahl der **verschiedenen** Weinregionen:

```
select count(distinct Region)
from ERZEUGER
```

- Weine, die älter als der Durchschnitt sind:

```
select Name, Jahrgang
from WEINE
where Jahrgang < (
    select avg(Jahrgang) from WEINE)
```

- Schachtelung von Aggregatfunktionen nicht erlaubt

```
select f1(f2(A)) as Ergebnis  
from R ...    -- (falsch!)
```

- mögliche Formulierung:

```
select f1(Temp) as Ergebnis  
from ( select f2(A) as Temp from R ...)
```

Aggregatfunktionen in where-Klausel

- Aggregatfunktionen liefern nur einen Wert \rightsquigarrow Einsatz in Konstanten-Selektionen der where-Klausel möglich
- alle Weingüter, die nur einen Wein liefern:

```
select * from ERZEUGER e
where 1 = (
    select count(*) from WEINE w
    where w.Weingut = e.Weingut)
```

group by und having

- Notation

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


Gruppierung: Schema

- Relation REL:

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

- Anfrage:

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

Gruppierung: Schritt 1

- from und where

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



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

Gruppierung: Schritt 2

- group by A, B

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



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

Gruppierung: Schritt 3

- `select A, sum(D)`

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



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

Gruppierung: Schritt 4

- having $A < 4$ and $\text{sum}(D) < 10$ and $\text{max}(C) = 4$

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



A	sum(D)
1	9

Gruppierung - Beispiel

- Anzahl der Rot- und Weißweine:

```
select Farbe, count(*) as Anzahl  
from WEINE  
group by Farbe
```

- Ergebnisrelation:

Farbe	Anzahl
Rot	5
Weiß	2

having - Beispiel

- Regionen mit mehr als einem Wein

```
select Region, count(*) as Anzahl  
from ERZEUGER natural join WEINE  
group by Region  
having count(*) > 1
```

Attribute für Aggregation bzw. having



- zulässige Attribute hinter select bei Gruppierung auf Relation mit Schema R
 - Gruppierungsattribute G
 - Aggregationen auf Nicht-Gruppierungsattributen $R - G$
- zulässige Attribute für having
 - dito

- zusätzlich zu klassischen Verbund (inner join): in SQL-92 auch äußerer Verbund \rightsquigarrow Übernahme von „dangling tuples“ in das Ergebnis und Auffüllen mit Nullwerten
- outer join übernimmt alle Tupel beider Operanden (Langfassung: full outer join)
- left outer join bzw. right outer join übernimmt alle Tupel des linken bzw. des rechten Operanden
- äußerer natürlicher Verbund jeweils mit Schlüsselwort natural, also z.B. natural left outer join

Äußere Verbunde /2

LINKS

A	B
1	2
2	3

RECHTS

B	C
3	4
4	5

NATURAL JOIN

A	B	C
2	3	4

OUTER

A	B	C
1	2	⊥
2	3	4
⊥	4	5

LEFT

A	B	C
1	2	⊥
2	3	4

RIGHT

A	B	C
2	3	4
⊥	4	5

Äußerer Verbund: Beispiel

```
select Anbaugebiet, count(WeinID) as Anzahl  
from ERZEUGER natural left outer join WEINE  
group by Anbaugebiet
```

Anbaugebiet	Anzahl
Barossa Valley	2
Napa Valley	3
Saint-Emilion	1
Pomerol	0
Rheingau	1

Simulation des (linken) äußeren Verbundes

```
select *  
from ERZEUGER natural join WEINE  
union all  
select ERZEUGER.*, cast(null as int),  
        cast(null as varchar(20)),  
        cast(null as varchar(10)),  
        cast(null as int),  
        cast(null as varchar(20))  
from ERZEUGER e  
where not exists (  
    select * from WEINE  
    where WEINE.Weingut = e.Weingut)
```

Sortierung mit order by

- Notation

order by *attributliste*

- Beispiel:

```
select *  
from WEINE  
order by Jahrgang
```

- Sortierung aufsteigend (asc) oder absteigend (desc)
- Sortierung als letzte Operation einer Anfrage \rightsquigarrow **Sortierattribut muss in der select-Klausel vorkommen**

- Sortierung auch mit berechneten Attributen (Aggregaten) als Sortierkriterium

```
select Weingut, count(*) as Anzahl  
from ERZEUGER natural join WEINE  
group by Weingut  
order by Anzahl desc
```

- Anfrage, die die **besten** k Elemente bzgl. einer Rangfunktion liefert

```
select w1.Name, count(*) as Rang
from WEINE w1, WEINE w2
where w1.Jahrgang <= w2.Jahrgang      -- Schritt 1
group by w1.Name, w1.WeinID          -- Schritt 2
having count(*) <= 4                  -- Schritt 3
order by Rang                         -- Schritt 4
```

Sortierung: Top-k-Anfragen

- Ermittlung der $k = 4$ jüngste Weine
- Erläuterung
 - Schritt 1: Zuordnung aller Weine die älter sind
 - Schritt 2: Gruppierung nach Namen, Berechnung des Rangs
 - Schritt 3: Beschränkung auf Ränge ≤ 4
 - Schritt 4: Sortierung nach Rang

Name	Rang
Zinfandel	1
Creek Shiraz	2
Chardonnay	3
Pinot Noir	4

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

Behandlung von Nullwerten /2

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

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

not	
true	false
unknown	unknown
false	true

- **Null-Selektion** wählt Tupel aus, die bei einem bestimmten Attribut Nullwerte enthalten

- Notation

attribut is null

- Beispiel

```
select * from ERZEUGER  
where Anbaubereich is null
```

Zusammenfassung



- SQL als Standardsprache
- SQL-Kern mit Bezug zur Relationenalgebra
- Erweiterungen: Gruppierung etc.

Kontrollfragen

- Welche Möglichkeiten der Formulierung von Verbunden gibt es?
- Was berechnen Aggregationen und Gruppierungen?
- Welche Operationen stehen für den Umgang mit Nullwerten zur Verfügung?

