

## Teil VI

# Die relationale Anfragesprache SQL

# Die relationale Anfragesprache SQL

- 1 Der SFW-Block in Detail
- 2 Erweiterungen des SFW-Blocks
- 3 Rekursion

# 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
```

# 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
```

# Natürlicher Verbund in SQL92

- 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

# Die `select`-Klausel

- Festlegung der Projektionsattribute

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

- mit

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

- ▶ 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
```

## 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  $\theta$  konstante

mögliche Vergleichssymbole  $\theta$  abhängig vom Wertebereich; etwa =, <>, >, <, >= sowie <=.

- ▶ Vergleich zwischen zwei Attributen mit kompatiblen Wertebereichen:

attribut1  $\theta$  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

- *Bereichsselektion*

*attrib* **between** *konstante*<sub>1</sub> **and** *konstante*<sub>2</sub>

ist Abkürzung für

*attrib*  $\geq$  *konstante*<sub>1</sub> **and**  
*attrib*  $\leq$  *konstante*<sub>2</sub>

- schränkt damit Attributwerte auf das abgeschlossene Intervall [*konstante*<sub>1</sub>, *konstante*<sub>2</sub>] ein
- Beispiel:

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

# Ungewissheitsselektion

- 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

- Beispiel

```
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 RoseZZZZZZZZZZZZZZZZ' ...
```

# Mengenoperationen

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

```
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
```

## Mengenoperationen in SQL /2

- 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

R union corresponding S	A	C
	1	3
	2	4
	2	3

R union corresponding by (A) S	A
	1
	2

# Schachtelung von Anfragen

- 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 Anbaugebiet = '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	<b>select distinct</b>
Selektion	<b>where</b> ohne Schachtelung
Verbund	<b>from, where</b> <b>from</b> mit <b>join</b> oder <b>natural join</b>
Umbenennung	<b>from</b> mit Tupelvariable; <b>as</b>
Differenz	<b>where</b> mit Schachtelung <b>except corresponding</b>
Durchschnitt	<b>where</b> mit Schachtelung <b>intersect corresponding</b>
Vereinigung	<b>union corresponding</b>

# Weiteres zu SQL

- Erweiterungen des SFW-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

# Skalare Ausdrücke

- Umbenennung von Spalten: *ausdruck* **as** *neuer-name*
- skalare Operationen auf
  - ▶ numerischen Wertebereichen: etwa +, −, \* und /,
  - ▶ Strings: Operationen wie **char\_length** (aktuelle Länge eines Strings), die Konkatenation || 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
- **Hinweise**:
  - ▶ skalare Ausdrücke können mehrere Attribute umfassen
  - ▶ Anwendung ist tupelweise: pro Eingabetupel entsteht ein Ergebnistupel

## Skalare Ausdrücke /2

- 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'
```

- 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
```

## Bedingte Ausdrücke

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

**case**

**when** *prädikat*<sub>1</sub> **then** *ausdruck*<sub>1</sub>

...

**when** *prädikat*<sub>*n*-1</sub> **then** *ausdruck*<sub>*n*-1</sub>

[ **else** *ausdruck*<sub>*n*</sub> ]

**end**

- 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

# Typkonvertierung

- explizite Konvertierung des Typs von Ausdrücken

```
cast(ausdruck as typname)
```

- Beispiel: **int**-Werte als Zeichenkette für Konkatenationsoperator

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

# Quantoren und Mengenvergleiche

- 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  $\theta$ -Vergleich mit *attribut* **true** wird
- **any** bzw. **some**: **where**-Bedingung wird erfüllt, wenn der  $\theta$ -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'
```

- 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)
```

- 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 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

- 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

## Aggregatfunktionen /2

- 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 \*

## Aggregatfunktionen /3

- 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)
```

- alle Weingüter, die nur einen Wein liefern:

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

## Aggregatfunktionen /2

- Schachtelung von Aggregatfunktionen nicht erlaubt

```
select  $f_1(f_2(A))$  as Ergebnis  
from  $R \dots$  -- (falsch!)
```

- mögliche Formulierung:

```
select  $f_1(\text{Temp})$  as Ergebnis  
from ( select  $f_2(A)$  as Temp from  $R \dots$  )
```

## 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

# Äußere Verbunde

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

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

# Simulation des äußeren Verbundes

- linker äußerer Verbund

```
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 /2

- 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
```

## Sortierung: Top-k-Anfragen

- 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
```

Name	Rang
Zinfandel	1
Creek Shiraz	2
Chardonnay	3
Pinot Noir	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  $\leq 4$
  - ▶ Schritt 4: Sortierung nach Rang

## Behandlung von Nullwerten

- 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

<b>and</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>true</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>unknown</b>	<b>unknown</b>	<b>unknown</b>	<b>false</b>
<b>false</b>	<b>false</b>	<b>false</b>	<b>false</b>

<b>or</b>	<b>true</b>	<b>unknown</b>	<b>false</b>
<b>true</b>	<b>true</b>	<b>true</b>	<b>true</b>
<b>unknown</b>	<b>true</b>	<b>unknown</b>	<b>unknown</b>
<b>false</b>	<b>true</b>	<b>unknown</b>	<b>false</b>

<b>not</b>	
<b>true</b>	<b>false</b>
<b>unknown</b>	<b>unknown</b>
<b>false</b>	<b>true</b>

## Selektionen nach Nullwerten

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

```
attribut is null
```

- Beispiel

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

## Benannte Anfragen

- Anfrageausdruck, der in der Anfrage mehrfach referenziert werden kann
- Notation

```
with anfrage-name [(spalten-liste) ] as  
( anfrage-ausdruck )
```

- Anfrage ohne **with**

```
select *  
from WEINE  
where Jahrgang - 2 >= (  
    select avg(Jahrgang) from WEINE)  
and Jahrgang + 2 <= (  
    select avg(Jahrgang) from WEINE)
```

## Benannte Anfragen /2

- Anfrage mit **with**

```
with ALTER(Durchschnitt) as (  
    select avg(Jahrgang) from WEINE)  
select *  
from WEINE, ALTER  
where Jahrgang - 2 >= Durchschnitt  
and Jahrgang + 2 <= Durchschnitt
```

# Rekursive Anfragen

- Anwendung: *Bill of Material*-Anfragen, Berechnung der **transitiven Hülle** (Flugverbindungen etc.)
- Beispiel:

<b>BUSLINIE</b>	<b>Abfahrt</b>	<b>Ankunft</b>	<b>Distanz</b>
	Nuriootpa	Penrice	7
	Nuriootpa	Tanunda	7
	Tanunda	Seppeltsfield	9
	Tanunda	Bethany	4
	Bethany	Lyndoch	14

## Rekursive Anfragen /2

- Busfahrten mit max. zweimalige Umsteigen

```
select Abfahrt, Ankunft
from BUSLINIE
where Abfahrt = 'Nuriootpa'
      union
select B1.Abfahrt, B2.Ankunft
from BUSLINIE B1, BUSLINIE B2
where B1.Abfahrt = 'Nuriootpa' and B1.Ankunft = B2.Abfahrt
      union
select B1.Abfahrt, B3.Ankunft
from BUSLINIE B1, BUSLINIE B2, BUSLINIE B3
where B1.Abfahrt = 'Nuriootpa' and B1.Ankunft = B2.Abfahrt
and B2.Ankunft = B3.Abfahrt
```

# Rekursion in SQL:2003

- Formulierung über erweiterte **with recursive**-Anfrage
- Notation

```
with recursive rekursive-tabelle as (  
    anfrage-ausdruck -- rekursiver Teil  
)  
[traversierungsklausel] [zyklus-klausel]  
anfrage-ausdruck -- nicht rekursiver Teil
```

- nicht rekursiver Teil: Anfrage auf Rekursionstabelle

## Rekursion in SQL:2003 /2

- rekursiver Teil:

-- Initialisierung

**select** ...

**from** *tabelle* **where** ...

-- Rekursionsschritt

**union all**

**select** ...

**from** *tabelle, rekursionstabelle*

**where** *rekursionsbedingung*

## Rekursion in SQL:2003: Beispiel

```
with recursive TOUR(Abfahrt, Ankunft) as (  
  select Abfahrt, Ankunft  
  from BUSLINIE  
  where Abfahrt = 'Nuriootpa'  
  union all  
  select T.Abfahrt, B.Ankunft  
  from TOUR T, BUSLINIE B  
  where T.Ankunft = B.Abfahrt)  
select distinct * from TOUR
```

# Schrittweiser Aufbau der Rekursionstabelle TOUR

## Initialisierung

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda

## Schritt 1

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany

## Schritt 2

Abfahrt	Ankunft
Nuriootpa	Penrice
Nuriootpa	Tanunda
Nuriootpa	Seppeltsfield
Nuriootpa	Bethany
Nuriootpa	Lyndoch

## Rekursion: Beispiel /2

- arithmetische Operationen im Rekursionsschritt

```
with recursive TOUR(Abfahrt, Ankunft, Strecke) as (  
  select Abfahrt, Ankunft, Distanz as Strecke  
  from BUSLINIE  
  where Abfahrt = 'Nuriootpa'  
  union all  
  select T.Abfahrt, B.Aankunft, Strecke + Distanz as Strecke  
  from TOUR T, BUSLINIE B  
  where T.Aankunft = B.Abfahrt)  
select distinct * from TOUR
```

# Sicherheit rekursiver Anfragen

- Sicherheit (= Endlichkeit der Berechnung) ist wichtige Anforderung an Anfragesprache
- Problem: Zyklen bei Rekursion

```
insert into BUSLINIE (Abfahrt, Ankunft, Distanz)  
values ('Lyndoch', 'Tanunda', 12)
```

- Behandlung in SQL
  - ▶ Begrenzung der Rekursionstiefe
  - ▶ Zyklenerkennung

## Sicherheit rekursiver Anfragen /2

- Einschränkung der Rekursionstiefe

```
with recursive TOUR(Abfahrt, Ankunft, Umsteigen) as (  
  select Abfahrt, Ankunft, 0  
  from BUSLINIE  
  where Abfahrt = 'Nuriootpa'  
    union all  
  select T.Abfahrt, B.Ankunft, Umsteigen + 1  
  from TOUR T, BUSLINIE B  
  where T.Ankunft = B.Abfahrt and Umsteigen < 2)  
select distinct * from TOUR
```

# Sicherheit durch Zyklenerkennung

- Zyklusklauseel

- ▶ beim Erkennen von Duplikaten im Berechnungspfad von *attrib*:  
Zyklus = '\*' (Pseudospalte vom Typ **char**(1))
- ▶ Sicherstellen der Endlichkeit des Ergebnisses „von Hand“

```
cycle attrib set marke to '*' default '-'
```

# Sicherheit durch Zyklenerkennung

```

with recursive TOUR(Abfahrt, Ankunft, Weg) as (
  select Abfahrt, Ankunft, Abfahrt || '-' || Ankunft as Weg
  from BUSLINIE where Abfahrt = 'Nuriootpa'
  union all
  select T.Abfahrt, B.Ankunft, Weg || '-' || B. Ankunft as Weg
  from TOUR T, BUSLINIE B where T.Ankunft = B.Abfahrt)
  cycle Ankunft set Zyklus to '*' default '-'
select Weg, Zyklus from TOUR

```

Weg	Zyklus
Nuriootpa-Penrice	-
Nuriootpa-Tanunda	-
Nuriootpa-Tanunda-Seppeltsfield	-
Nuriootpa-Tanunda-Bethany	-
Nuriootpa-Tanunda-Bethany-Lyndoch	-
Nuriootpa-Tanunda-Bethany-Lyndoch-Tanunda	*

# SQL-Versionen

- Geschichte
  - ▶ SEQUEL (1974, IBM Research Labs San Jose)
  - ▶ SEQUEL2 (1976, IBM Research Labs San Jose)
  - ▶ SQL (1982, IBM)
  - ▶ ANSI-SQL (SQL-86; 1986)
  - ▶ ISO-SQL (SQL-89; 1989; drei Sprachen Level 1, Level 2, + IEF)
  - ▶ (ANSI / ISO) SQL2 (als SQL-92 verabschiedet)
  - ▶ (ANSI / ISO) SQL3 (als SQL:1999 verabschiedet)
  - ▶ (ANSI / ISO) SQL:2003
- trotz Standardisierung: teilweise Inkompatibilitäten zwischen Systemen der einzelnen Hersteller

# Zusammenfassung

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