



Kapitel 3 – Einleitung zu SQL

Vorlesung Datenbanken

Wintersemester 2017/18

Dr. Kai Höfig



- ◆ **3.1 SQL-DDL Datendefinitionsoperationen**
- ◆ 3.2 SQL-DML Änderungsoperationen
- ◆ 3.3 SQL-DML Anfrageoperationen



SQL = Structured Query Language

- Relativ einfach aufgebaute Sprache
- Angelehnt an englischen Umgangssprache (viel „syntaktischer Zucker“)

SQL-DDL: Data Definition Language

Manipulation der Tabellenschemata

- Erzeugen/Ändern/Löschen von
 - Tabellenschemas
 - Datenbanken
 - Sichten
 - Indexen

SQL-DML: Data Manipulation Language

Manipulation der Tupel (Datensätze)

- **Änderungsoperationen:** Tupel
Einfügen/Ändern/Löschen
- **Anfrageoperationen:**
 - Gruppieren & Sortieren
 - Abfragen von Daten
 - Transaktionen



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
- SQL/XML:2006 – fügt XML Behandlung hinzu
- SQL:2008
- SQL:2011 aktuellste Version (noch wenig verbreitet)

♦ Jedes DBMS implementiert SQL mit unterschiedlichen Details und Erweiterungen

- ➔ Rückgriff auf die Dokumentation immer nötig (online verfügbar)
- ➔ In Vorlesung: SQL:2008 (aktueller SQL Standard)
- ➔ In Übung: Transact-SQL (SQL Variante des Microsoft SQL Servers)



Praktische Bedeutung von SQL

- ◆ Vielfältiger **Einsatz von SQL** in der Wirtschaftsinformatik, u.a.
 - Datenbankentwicklung (Tabellen, Sichten, Rechte etc. erstellen & warten.)
 - Applikationsentwicklung (Daten manipulieren und präsentieren).
 - Erstellung von Internetauftritten (dynamischer Web-Seiten, meist Zugriff mittels Skriptsprachen wie JavaScript, ASP.NET, PHP, o.ä.)
 - Mobile Applikationen (iOS, Android, etc.)
 - Data Warehouses / Business Intelligence Systeme
 - Betriebliches Informationsmanagement insb. in ERP Systemen wie SAP
 - usw.

- ◆ Häufig jedoch **unterschiedliche, simultane Zugriffswege** auf die DB
 - DB Verwaltungstools (wie das SQL Server Management Studio) zum Anlegen von DBs, Erteilen/Entziehen von Zugriffsrechten, Backup, Optimierung etc.
 - Zugriff via SQL aus den Anwendungen



SQL-DDL – Wichtige Anweisungen

♦ **create table**

- **Anlegen** einer neuen (leeren) Relation
- Festlegen der Integritätsbedingungen
- Ablegen der Informationen im Data Dictionary

♦ **drop table**

- **Löschen** einer (leeren) Relation
- Löschen der Informationen aus dem Data Dictionary

♦ **alter table**

- **Hinzufügen/Löschen** von Attribute/Integritätsbedingungen einer bestehenden Relation
- Update der Informationen im Data Dictionary
- Siehe Übung!



Beispiel für `create table`

- ♦ Anlegen der bekannten ERZEUGER Relation:

```
create table ERZEUGER(  
    Weingut varchar(50),  
    Anbaugebiet varchar(20) not null,  
    Region varchar(10),  
    primary key (Weingut))
```



Mögliche Wertebereiche in SQL

- ♦ `integer` (oder auch `integer4`, `int`),
- ♦ `smallint` (oder auch `integer2`),
- ♦ `float(p)` (oder auch kurz `float`),
- ♦ `decimal(p,q)` und `numeric(p,q)` mit jeweils `q` Nachkommastellen,
- ♦ `character(n)` (kurz `char(n)`, bei `n = 1` auch `char`) für Zeichenketten (Strings) fester Länge `n`,
- ♦ `character varying(n)` (kurz `varchar(n)`) für Strings variabler Länge bis zur Maximallänge `n`,
- ♦ `bit(n)` oder `bit varying(n)` analog für Bitfolgen, und
- ♦ `date`, `time` bzw. `timestamp` für Datums-, Zeit- und kombinierte Datums-Zeit-Angaben
- ♦ ...plus typischerweise einige mehr ja nach DBMS!



Schlüsselbedingungen in SQL

- ♦ **primary key** kennzeichnet Spalte/n als **primäres Schlüsselattribut**
- ♦ **unique** kennzeichnet Spalte/n als **(nicht primäres) Schlüsselattribut**
- ♦ Falls Schlüssel aus nur einem Attribut besteht: direkt hinter Attribut möglich

```
create table ERZEUGER(  
    Weingut varchar(50) primary key,  
    Anbaugebiet varchar(20) not null,  
    Region varchar(10))
```



Nullwerte in SQL

◆ Spezieller Wert `null`

- repräsentiert die Bedeutung „Wert unbekannt“, „Wert nicht anwendbar“ oder „Wert existiert nicht“, gehört aber zu keinem Wertebereich
- Kennzeichnung von Nullwerte in SQL durch `null` oder \perp

◆ `null` kann in allen Spalten auftauchen, außer

- in primären Schlüsselattributen und
- den mit `not null` gekennzeichneten

◆ `create table`

- `not null` schließt in bestimmten Spalten **Nullwerte** als Attributwerte aus
- `null` erlaubt in der Spalte Nullwerte (selten nötig)
- Bem: auch bei `alter table` anwendbar



Weiteres zur Datendefinition in SQL

- ◆ Neben Primär- und Fremdschlüsseln können in SQL angegeben werden:
 - mit der **default**-Klausel: Defaultwerte für Attribute,
 - mit der **create domain**-Anweisung: benutzerdefinierte Wertebereiche und
 - mit der **check**-Klausel: weitere lokale Integritätsbedingungen innerhalb der zu definierenden Wertebereiche, Attribute und Relationenschemata
- ➔ Details siehe SQL-Dokumentation



drop table

- ◆ Beispiel: Löschen der WEINE Relation

```
drop table WEINE
```

- ◆ Bedingung: Auf die Relation dürfen keine referentiellen Integritätsbedingungen verweisen



- ◆ Einführung in
 - TSQL,
 - MS SQL Server 2016
 - MS SQL Server Management Studio
- ◆ Create table, drop table, alter table für ein Beispielszenario:
- ◆ Cocktails
 - Welche Tabellen brauchen wir?
 - Welche Attribute benötigen wir?
 - Was sind Keys?



- ◆ 3.1 SQL-DDL Datendefinitionsoperationen
- ◆ **3.2 SQL-DML Änderungsoperationen**
- ◆ 3.3 SQL-DML Anfrageoperationen



Änderungsoperationen in SQL

- ◆ **insert**

Einfügen eines oder mehrerer Tupel in eine Basisrelation oder Sicht

- ◆ **update**

Ändern von einem oder mehreren Tupel in einer Basisrelation oder Sicht

- ◆ **delete**

Löschen eines oder mehrerer Tupel aus einer Basisrelation oder Sicht

- ◆ Integritätsbedingungen (Schlüsselbedingungen, Referentielle Integrität, etc.). automatisch vom DBMS überprüft

- Falls durch Befehl verletzt: Fehlermeldung, Befehl nicht ausgeführt

- ◆ Mit **select** auch **CRUD Operationen** genannt
(Create, Read, Update, Delete)



Die `insert`-Anweisung

♦ Syntax

```
insert into relation [ (attribut1, ..., attributn) ]  
values (konstante1, ..., konstanten)
```

- optionale Attributliste ermöglicht das Einfügen von unvollständigen Tupeln

♦ Beispiel

```
insert into ERZEUGER  
values ('Chateau Lafitte', 'Medoc', 'Bordeaux')
```

♦ Nicht alle Attribute angegeben → fehlenden Attribute werden null/default

```
insert into ERZEUGER (Weingut, Region)  
values ('Wairau Hills', 'Marlborough')
```


insert: Einfügen von berechneten Daten

◆ Syntax:

```
insert into relation [ (attribut1, ..., attributn) ]  
      SQL-Anfrage
```

◆ Beispiel:

```
insert into WEINE  
      select ProdID, ProdName, 'Rot', ProdJahr, 'Chateau Lafitte'  
      from LIEFERANT  
      where LName = 'WeinKontor'
```



Die delete-Anweisung

- ◆ Syntax:

```
delete from relation  
[ where bedingung ]
```

- ◆ Beispiel: Löschen eines Tupels in der WEINE-Relation:

```
delete from WEINE  
where WeinID = 4711
```



Weiteres zu delete

- ◆ Standardfall ist das Löschen mehrerer Tupel:

```
delete from WEINE  
where Farbe = 'Weiß'
```

- ◆ Löschen der gesamten Relation:

```
delete from WEINE
```

- ◆ Löschoperationen können zur Verletzung von Integritätsbedingungen führen!
 - Beispiel: Verletzung der Fremdschlüsseigenschaft, falls es noch Weine von diesem Erzeuger gibt:

```
delete from ERZEUGER  
where Region = 'Hessen'
```



Die update-Anweisung

♦ Syntax

```
update relation  
set    attribut1 = ausdruck1  
        ...  
        attributn = ausdruckn  
[ where bedingung ]
```



Beispiel für update

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut	Preis
	3456	Zinfandel	Rot	2004	Helena	5,99
	2171	Pinot Noir	Rot	2001	Creek	10,99
	3478	Pinot Noir	Rot	1999	Helena	19,99
	4711	Riesling Reserve	Weiß	1999	Müller	14,99
	4961	Chardonnay	Weiß	2002	Bighorn	9,90

```
update WEINE
set      Preis = Preis * 1.10
where    Jahrgang < 2000
```

WEINE	WeinID	Name	Farbe	Jahrgang	Weingut	Preis
	3456	Zinfandel	Rot	2004	Helena	5,99
	2171	Pinot Noir	Rot	2001	Creek	10,99
	3478	Pinot Noir	Rot	1999	Helena	21,99
	4711	Riesling Reserve	Weiß	1999	Müller	16,49
	4961	Chardonnay	Weiß	2002	Bighorn	9,90



Weiteres zu **update**

- ◆ Realisierung von Eintupel-Operation mittels (Primär-)Schlüssel:

```
update   WEINE  
set      Preis = 7.99  
where    WeinID = 3456
```

- ◆ Vorsicht: ohne `where` Bedingung: Änderung der gesamten Relation:

```
update   WEINE  
set      Preis = 11
```



SQL-DML – praktische Umsetzung der Relationalen Algebra

- ◆ Anfragen an relationales DBMS werden nicht direkt in relationaler Algebra gestellt
- ◆ Sondern sind als Teil der SQL-DML umgesetzt
- ◆ SQL implementiert die Operationen der relationalen Algebra relativ direkt
- ◆ Jedoch mit erheblichem syntaktischen Zucker
- ◆ SQL-DML besteht somit aus
 - Änderungsoperationen: insert/update/delete Statements
 - Anfrageoperationen der rel. Algebra: select/union/except/intersect Statements



SQL-Kern – der SFW-Block

- ◆ **select** Projektionsliste
arithmetische Operationen & Aggregatfunktionen
- ◆ **from** zu verwendende Relationen, evtl. Umbenennungen
- ◆ **where** Selektions-, Verbundbedingungen
geschachtelte Anfragen (wieder ein SFW-Block)
- ◆ **group by** Gruppierung für Aggregatfunktionen
- ◆ **having** Selektionsbedingungen an Gruppen
- ◆ **order by** Ausgabereihenfolge

hier
Kapitel "Advanced SQL"



Projektion π in SQL

- ◆ Ausdruck in Relationenalgebra

$$\pi_{\text{WeinID, Weingut}}(\text{WEINE})$$

- ◆ Anfragen in SQL

```
select  WEINE.WeinID, WEINE.Weingut  
from    WEINE
```

```
select  WeinID, Weingut  
from    WEINE
```



Selektion σ in SQL

♦ Ausdruck in Relationenalgebra

$\sigma_{\text{Jahrgang} > 2000}(\text{WEINE})$

♦ Anfrage in SQL

```
select  *  
from    WEINE  
where   WEINE.Jahrgang > 2000
```

```
select  *  
from    WEINE  
where   Jahrgang > 2000
```



Kombination von Selektion und Projektion

Anfrage an eine einzelne Tabelle mit Selektion und Projektion

- ◆ Ausdruck in Relationenalgebra

$$\pi_{\text{Name, Farbe}}(\sigma_{\text{Jahrgang}=2002}(\text{WEINE}))$$

- ◆ Anfrage in SQL

```
select  WEINE.Name, WEINE.Farbe  
from    WEINE  
where   WEINE.Jahrgang = 2002
```

```
select  Name, Farbe  
from    WEINE  
where   Jahrgang = 2002
```



Wichtigster Unterschied zwischen SQL und Relationaler Algebra

Relationale Algebra hat immer **Mengensemantik**

→ Duplikate im Ergebnis werden automatisch entfernt!

SQL hat standardmäßig **Multimengensemantik**

→ Duplikate im Ergebnis werden **nicht** automatisch entfernt!

- ◆ Grund: Performance! Duplikat Entfernung ist teuer: $O(n \log n)$
- ◆ Explizite Mengensemantik in SQL durch **distinct**



Multimengensemantik von SQL - Beispiel

- ◆ Ausdruck in Relationaler Algebra:

$\pi_{\text{Region}}(\text{ERZEUGER})$

- ◆ Anfrage(n) in SQL

```
select Region  
from ERZEUGER
```

Region
South Australia
Kalifornien
Bordeaux
Bordeaux
Hessen
Kalifornien

\Downarrow

```
select distinct Region  
from ERZEUGER
```

Region
South Australia
Kalifornien
Bordeaux
Hessen

- ◆ Multimengensemantik in Relationaler Algebra nicht möglich!



Kreuzprodukt \times in SQL

- ◆ Ausdruck in Relationenalgebra

WEINE \times FLASCHE

- ◆ Mögliche Anfragen in SQL

```
select  *  
from    WEINE, FLASCHE
```

```
select  *  
from    WEINE cross join FLASCHE
```



Natürlicher Verbund \bowtie in SQL

- ♦ Ausdruck in Relationenalgebra : $WEINE \bowtie ERZEUGER$
- ♦ Mögliche Anfragen für den natürlichen Verbund in SQL

1. Verwendung von **natural join**

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

2. Explizite **Verbundbedingung**

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

3. Verwendung von **using**

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

4. Verwendung von **join on**

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



Kombination von Operationen

- ◆ Ausdruck in Relationenalgebra

$\pi_{\text{Name, Farbe, Weingut}}(\sigma_{\text{Jahrgang} > 2000}(\text{WEINE}) \bowtie \sigma_{\text{Region} = \text{„Kalifornien“}}(\text{ERZEUGER}))$

- ◆ 2 Möglichkeiten für diese Anfrage in SQL (es gibt noch viele weitere!)

```
select  Name, Farbe, WEINE.Weingut
from    WEINE join ERZEUGER on WEINE.Weingut = ERZEUGER.Weingut
where    Jahrgang > 2000 and
          Region = 'Kalifornien'
```

```
select  Name, Farbe, WEINE.Weingut
from    WEINE, ERZEUGER
where    Jahrgang > 2000 and
          Region = 'Kalifornien' and
          WEINE.Weingut = ERZEUGER.Weingut
```




Die `from`-Klausel

Die `from`-Klausel

- ◆ Einfachste Form

```
select *  
from    relation1, relation2, ..., relationn
```

- ◆ Beispiel:

```
select *  
from    WEINE
```

- ◆ Bei mehr als einer Relation wird das kartesische Produkt gebildet.

- Beispiel:

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



from : Tupelvariablen für mehrfachen Zugriff

- ◆ Einführung von Tupelvariablen erlaubt mehrfachen Zugriff auf eine Relation.
 - Hinter jeder Relation kann optional eine Tupelvariable stehen (und dazwischen optional **as**)
- ◆ Beispiel:

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

```
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



from : Tupelvariable für Zwischenergebnisse

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

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

- ♦ **as** ist erneut optional



Die `select`-Klausel

- ◆ Festlegung der Projektionsattribute

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

- ◆ Mit
projektionsliste := { *attribut* | *arithmetischer Ausdruck* | *Aggregatfunktion* | * } [, ...]
- ◆ Und
 - *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



select: Projektionsliste *

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

- ◆ Beispiel

```
select *  
from WEINE
```



`select: distinct` eliminiert Duplikate

- ◆ `distinct` eliminiert Duplikate
- ◆ Beispiel:

```
select Name  
from WEINE
```

→ liefert Multimenge

Name
La Rose GrandCru
Creek Shiraz
Zinfandel
Pinot Noir
Pinot Noir
Riesling Reserve
Chardonnay

```
select distinct Name  
from WEINE
```

→ liefert Menge

Name
La Rose GrandCru
Creek Shiraz
Zinfandel
Pinot Noir
Riesling Reserve
Chardonnay



select: Tupelvariablen und Relationennamen

◆ Anfrage

```
select Name  
from WEINE
```

◆ ist äquivalent zu

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

◆ und

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



select: Präfixe für Eindeutigkeit

- ◆ Falsches Beispiel:

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



- ◆ Attribut `Weingut` existiert sowohl in Tabelle `WEINE` als auch in `ERZEUGER`!

- ◆ Richtiges Beispiel mit Präfix:

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




select: Tupelvariablen für Eindeutigkeit

- ◆ Bei der Verwendung von Tupelvariablen, kann der Name einer Tupelvariablen zur Qualifizierung eines Attributs benutzt werden.
- ◆ Beispiel:

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



select/where: Skalare Ausdrücke

Skalare Operationen

- ◆ Numerischen Wertebereiche: etwa +, -, * und /
- ◆ Strings: Typische String-Operationen wie z.B.
 - `char_length(str)`: aktuelle Länge eines Strings
 - `str1 || str2`: Konkatenation der Strings str1 und str2
 - `substring(str, start, len)`: Teilzeichenkette
 - `position(str1 in str2)`: Position des ersten Auftretens von str1 in str2 (≥ 1 ; 0 wenn nicht enthalten)
- ◆ Datumstypen & Zeitintervallen: Operationen wie +, -, *; Funktionen wie z.B.
 - `current_date`: aktuelles Datum
 - `current_time`: aktuelle Zeit
 - `year(d)`, `month(d)`, `day(d)`: Jahr, Monat, Tag eines Datums
- ◆ Hinweis
 - Ausdrücke können mehrere Attribute umfassen
 - Anwendung ist tupelweise: pro Eingabetupel entsteht ein Ergebnistupel



select/where: Skalare Ausdrücke - Beispiele

- ♦ Ausgabe der Namen aller Grand Cru-Weine

```
select substring(Name from 1 for  
           position('GrandCru' in Name) - 2)  
from      WEINE where Name like '%GrandCru'
```

- ♦ 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
```



select/where: Bedingte Ausdrücke

- ♦ **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
```

- ♦ 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
```



select/where: Typkonvertierung

- ◆ Explizite Konvertierung des Typs von Ausdrücken mit **cast**

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

- ◆ Beispiel: int-Werte als Zeichenkette für Konkatenationsoperator

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



Die where-Klausel

♦ 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,
z.B. =, <>, >, <, >= sowie <=.
- Vergleich zwischen zwei Attributen mit kompatiblen Wertebereichen:
attribut1 θ *attribut2*
- logische Konnektoren **or**, **and** und **not**



where: Verbundbedingung

- ♦ Verbundbedingung hat die Form:

```
relation1.attribut = relation2.attribut
```

- ♦ Beispiel:

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



where: Bereichsselektion

- ◆ 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
```




where: Ungewissheitsselektion (1)

- ◆ 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



where: 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' ...
```



where: Quantoren und Mengenvergleiche

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

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

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



where: 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')
```



where: 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'
```



Umbenennung β in SQL

- ♦ Ausdruck in Relationenalgebra

$$\beta_{\text{Name} \leftarrow \text{Wein}}(\text{EMPFEHLUNG})$$

- ♦ Mögliche Anfragen in SQL

```
select  EMPFEHLUNG.Wein as Name  
from    EMPFEHLUNG
```

```
select  Wein as Name  
from    EMPFEHLUNG
```

Bem: **as** ist optional, wird aber i.allg. verwendet



Mengenoperationen \cup , $-$, \cap in SQL

- ♦ Vereinigung = **union**, Differenz = **except**, Schnittmenge = **intersect**

```
select  Nachname
from    WINZER
union
select  Nachname
from    KRITIKER
```

```
select  Nachname
from    WINZER
except
select  Nachname
from    KRITIKER
```

```
select  Nachname
from    WINZER
intersect
select  Nachname
from    KRITIKER
```



Mengenoperationen

- ◆ Mengenoperationen werden nach Position der Attribute ausgeführt, nicht nach Namen der Attribute und erfordern kompatible Wertebereiche
 - 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
```

- ◆ bei Vereinigung
 - Default-Fall ist Duplikateliminierung (**union distinct**)
 - ohne Duplikateliminierung durch **union all**

◆ Notation:

```
attribut [not] in ( SFWblock )
```

◆ Beispiel:

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



in: (interne) Auswertung von geschachtelten Anfragen

- ♦ Auswertung der inneren Anfrage zu den Weingütern aus Bordeaux
- ♦ Einsetzen des Ergebnisses als Menge von Konstanten in die äußere Anfrage hinter **in**
- ♦ Auswertung der modifizierten Anfrage

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

Name
La Rose Grand Cru



in: 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 )
```



Das **exists/not exists**-Prädikat

- ◆ Einfache Form der Schachtelung mittels **(not) exists**

```
exists ( SFWblock )
```

- ◆ liefert **true**, wenn das Ergebnis der inneren Anfrage nicht **leer** ist
(**not exists** liefert natürlich **true**, wenn es leer ist)
- ◆ Beispiel: Weingüter aus Bordeaux ohne gespeicherte Weine:

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



Verzahnt geschachtelte Anfragen

- ◆ `in/exists` häufig bei **verzahnt geschachtelten** (korrelierte) **Anfragen**
 - D.h. in der inneren Anfrage wird Relationen- oder Tupelvariablen-Name aus dem **from**-Teil der äußeren Anfrage verwendet

- ◆ Beispiel: Weingüter mit 1999er Rotwein

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

- Konzeptionelle Auswertung
 - Untersuchung des ersten ERZEUGER-Tupels in der äußeren Anfrage und Einsetzen in innere Anfrage
 - Auswertung der inneren Anfrage
 - Weiter bei mit zweitem Tupel . . .



Verzahnt geschachtelte Anfragen – Umformung in Verbund

- ◆ Verzahnt geschachtelte Anfragen können durch joins ersetzt werden
 - Verzahnt geschachtelte Anfragen daher i.allg. vermeiden, da unübersichtlich!
- ◆ Beispiel: Weingüter mit 1999er Rotwein

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

- ◆ Alternative Formulierung als Verbund

```
select ERZEUGER.*  
from ERZEUGER natural join WEINE  
where Jahrgang=1999 and Farbe='Rot'
```



Mächtigkeit des SQL-Kerns

Relationenalgebra	SQL
Projektion	<code>select distinct</code>
Selektion	<code>where</code> ohne Schachtelung
Kreuzproduct	<code>cross join</code> oder „Komma“
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</code>
Durchschnitt	<code>where</code> mit Schachtelung <code>intersect</code>
Vereinigung	<code>union</code>



Äquivalente Anfragen und Anfrageoptimierung

- ◆ Viele Anfragen in SQL bzw. der Relationalen Algebra haben äquivalente Anfragen, d.h. Anfragen, die dasselbe Ergebnis liefern
 - In den Übungen werden Sie viele Beispiele kennen lernen (bzw. selbst finden)
- ◆ Manche Anfragen deutlich effizienter auswertbar, als (äquivalente) andere
- ◆ Vorteil der Relationalen Algebra: Ausdrücke können (mittels mathematischer Regeln) umgeformt werden, wobei die Äquivalenz garantiert ist
- ◆ **Query Optimizer** (Teil des Query Processors)
 - Bestandteil jedes DBMS
 - Aufgabe: Umformen jeder SQL Anfragen (i.allg. nach vorhergehender Transformation in die Relationale Algebra) in einen möglichst effizient ausführbaren äquivalenten Ausdruck
 - Gehören zu den komplexesten Softwaremodulen die es überhaupt gibt!