



3. SQL – Die Anfragesprache für relationale Datenbanken

■ SQL = structured query language

■ 1974

- Entwicklung der Sprache durch Don Chamberlin bei der Firma IBM
- Prototypische Implementierung in System/R

■ 1981

- Erste kommerzielle Produkte von Oracle und IBM



■ Marktbeherrschende Sprache im Datenbankbereich

- Mischung zwischen Tupelkalkül und relationaler Algebra
- Unterstützung von Mengen- und Multimengensemantik
- Unterstützung von Aggregation, Gruppieren und Sortieren



Geschichte von SQL

- **SQL** (Structured **Q**uery **L**anguage) ist heute die populärste und verbreitetste relationale Datenbanksprache. Jedes relationale DBMS "versteht,, SQL!
- SQL wurde Anfang der 1970er Jahre bei **IBM** entwickelt (als Sprache für das relationale Prototyp-DBMS "System R"). Interessante Quelle zur „Geschichte“ von SQL: „The 1995 SQL Reunion“ (s. Literaturwebseite dieser Vorlesung)
- ursprüngliche Bezeichnung: **SEQUEL** (Structured **E**nglish Query Language)
- **erste SQL-Normung** (SQL 1) 1986 durch ANSI in den USA, revidiert 1989
- wesentliche Erweiterung 1992: **SQL2** bzw. SQL92 (standardisiert, heute üblich)
- 1999, 2003, 2006 und 2008 jeweils neue **Standards**: **SQL:1999**, ... , **SQL:2016** (Neuerungen sind oft erst ansatzweise in kommerziellen Systemen zu finden.)
- **Vorsicht** ! Nahezu jedes kommerzielle DB-Produkt hat seinen eigenen "**Dialekt**" von SQL, der mehr oder weniger gut mit dem Standard kompatibel ist!



Standardisierung von SQL

■ Historie

- SQL wurde bereits 1986 erstmals standardisiert.
- SQL92 (aus dem Jahre 1992) ist der bekannteste Standard
 - Unterstützung zwischen Grad der Standardisierung:
 - Entry, Intermediate, Full
 - Entry-Level wird heute von nahezu allen Systemen unterstützt.
- SQL 1999
 - Substantielle Erweiterungen von SQL
 - Rekursion, Prozedurale Erweiterungen
- SQL 2006
 - Unterstützung von XML
- SQL 2011
 - Unterstützung von Zeit (Anwendungszeit, Systemzeit)
- SQL 2016
 - Unterstützung von JSON als Datentyp
 - Pattern-Matching auf Relationen mit temporalen Attribut



Überblick zu SQL

- **SQL besteht aus folgenden Teilsprachen**
 - DDL (Data Definition Language)
 - Erzeugen von Datenstrukturen wie z. B.
 - Datenbanken, Relationen, Indexe, Sichten
 - DML (Data Manipulation Language)
 - Einfügen, Löschen, Ändern von Daten
 - Anfragen

- **Konzeption von SQL**
 - Zunächst war SQL eine interaktive Sprache.
 - Heute wird SQL oft als eine eingebettete Anfragesprache in den üblichen Sprachen benutzt.
 - aufrufbar aus Java, C++, C#, Scala, ...



Wichtige Syntaxregeln

- **SQL unterscheidet nicht zwischen Groß- und Kleinschreibung („case insensitive“)**
 - Schlüsselwörter und Bezeichner
- **Bezeichner in SQL**
 - Namen von Datenbanken, Relationen, Attributen, Integritätsbedingungen, Indexe
 - Syntaktische Konventionen, wie üblich
 - erstes Zeichen ist ein Buchstabe,
 - weiterhin können darin Zahlen und _ enthalten sein,
 - Bezeichner müssen sich von einem Schlüsselwort unterscheiden.
- **SQL-Befehle enden mit einem Semikolon: ;**
- **Kommentare in SQL beginnen mit zwei Strichen: --**



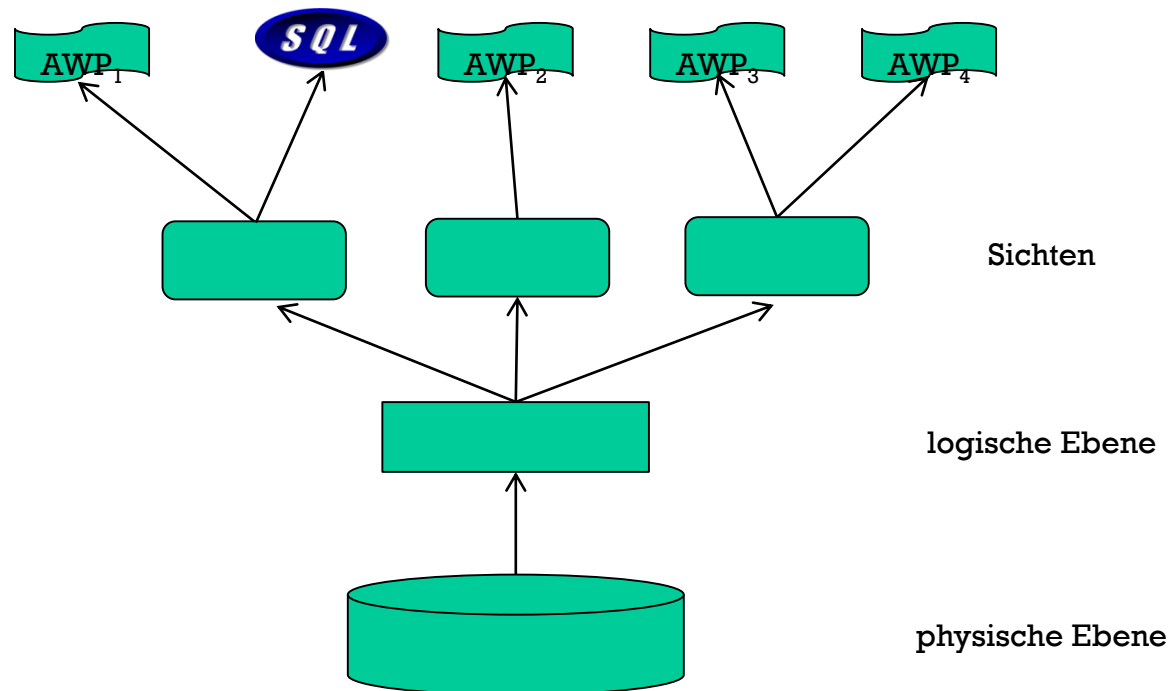
3.1 Datendefinitionssprache

- Anlegen einer Datenbank
- Anlegen und Ändern von Relationen in einer Datenbank
- Zudem kann man noch weitere Strukturen wie z. B. Views und Indexe anlegen (Details dazu später)

create view

create table

create index





Kurzer Überblick zur DDL

- Im Folgenden soll schnell ein Einstieg in die DDL gegeben werden, um anschließend in die DML einzuführen.

- **Anlegen einer Datenbank**

create database <name>

- Eine neue Datenbank wird erstellt und angelegt.

- **Anlegen einer Relation**

create table <relname> (
(<Relationenkomponente>[,<Relationenkomponente>]·)

- <Relationenkomponente>
 - Deklaration eines Attributs oder einer Integritätsbedingung



Anlegen der ERP-Datenbank (1)

■ Beispiel

■ create table Maschinen mnr int primary key,
mname varchar(10));

Relationenname

Attributname

Typ

Integritätsbedingung
Primärschlüssel
(ohne Namen)

■ Achtung!

■ Bei der Attributdeklaration

- Erst der Name, dann der Typ!
- Optional folgt noch eine Integritätsbedingung

■ create table Abteilung(abtnr int primary key,
aname varchar(10));



Anlegen der ERP-Datebank (2)

- create table Personal(pnr int primary key,
pname varchar(10) not null,
vorname varchar(10),
abtnr int references Abteilung(abtnr),
lohn int default 20000);

Fremdschlüsselbedingung
auf das Attribut abtnr der
Relation Abteilung

Defaultwert des Attributs

- create table leitet(
pnr int primary key references Personal(pnr),
abtnr int references Abteilung(abtnr));



Anlegen der ERP Datenbank (3)

- create table Abteilung(abtnr int primary key,
 aname varchar(10));
- create table pmzuteilung(
 pnr int references Personal(pnr),
 mnr int references Maschinen(mnr),
 note int,
 constraint pk primary key (pnr,mnr));

Separate Integritätsbedingung mit Bezeichner pk

- Dies ist erforderlich, da Primärschlüssel aus zwei Attributen besteht.



Löschen von Strukturen

■ Löschen

- drop table <name>
- Entsprechend für alle anderen Strukturen
 - drop database <name>

■ Ändern/Löschen eines Relationenschemas

- alter table <Relationen-Name>
add <Relationenkomponente>



Verwaltung der Strukturen in einem Datenbankkatalog

pgAdmin III

Datei Bearbeiten Plugins Anzeigen Werkzeuge Hilfe

Objektbrowser

PostgreSQL 9.1 (localhost:5432)

- Datenbanken (2)
 - ERP
 - Kataloge (2)
 - Extensions (1)
 - Schemata (1)
 - public
 - Sortierfolgen (0)
 - Domänen (0)
 - Volltextsuche - Konfigurationen (0)
 - Volltextsuche - Wörterbücher (0)
 - Volltextsuche - Parser (0)
 - Volltextsuche - Vorlagen (0)
 - Funktionen (0)
 - Sequenzen (0)
 - Tabellen (5)
 - abteilung
 - leitet
 - maschinen
 - personal
 - pmzuteilung
 - Triggerfunktionen (0)
 - Sichten (1)

Eigenschaften

Tabelle	Eigentümer
abteilung	postgres
leitet	postgres
maschinen	postgres
personal	postgres
pmzuteilung	postgres

SQL-Feld

Hole Details zu Tabellen... Fertig. 0,00 Sek.



3.2 Datenmanipulationssprache (DML)

■ Funktionalität

- Einfügen, Ändern und Löschen von Tupeln
 - Operationen auf einem Tupel
 - Massenoperationen
- Anfragen auf Relationen
 - Mischform zwischen Relationaler Algebra und Tupelkalkül
 - Deklarative Formulierung
 - Übersetzung der Anfrage in einen Operatorbaum



Einfügen von Tupeln in eine Relation

- Einfachste Form beim Einfügen
insert into <Relationen-Name>
[(<Attributname> [, <Attributname>]*)]
values (<Konstante> [, <Konstante>]*)
- Beispiel
insert into PMZuteilung values (51, 84, 2);
- Einfügen von mehreren Datensätzen möglich:
insert into PMZuteilung values (51, 84, 2), (82, 101, 2);



NULL-Werte

■ Problem

- Es soll in die Relation PMZuteilung eingetragen werden, dass der Angestellte mit pnr = 82 die Maschine mit mnr = 84 bedienen kann.
- Leider ist die Note des Angestellten für die Maschine noch nicht bekannt.

■ Lösung

- Um das Einfügen unvollständiger Datensätze zu unterstützen, bekommen die nicht mit einem Wert belegten Attribute den Wert NULL zugewiesen.
- Somit ist der Befehl
insert into PMZuteilung (pnr, mnr) values (82,84)
möglich.
 - Dabei müssen jetzt die **Attribute** noch angegeben werden.



Attribute ohne NULL-Werte

- Der Wert NULL darf keinem Primärschlüssel einer Relation zugewiesen werden.
- Zudem kann durch die Integritätsbedingung not null verhindert werden, dass ein Attribut **einen Null-Wert** bekommt.
- Eine weitere Möglichkeit ist die Verwendung eines Default-Werts bei der Definition des Schemas.
 - `create table personal (...,
 lohn int default 20000);`



Sicherstellung der Integritätsbedingungen

- **Nach dem Einfügen wird i. A. überprüft, ob die Integritätsbedingung noch erfüllt sind.**
 - Wenn nicht, wird die Operation rückgängig gemacht und ein Fehler gemeldet.
- **Art der Überprüfungen**
 - Primärschlüssel
 - Testen, ob ein gleicher Schlüsselwert bereits in der Datenbank vorhanden ist.
 - Fremdschlüssel
 - Testen, ob der Wert bereits in der Relation vorhanden ist, wo das entsprechende Attribut als Primärschlüssel vorliegt.
- **Wir werden im Kapitel über Transaktionen noch genauer auf diesen Sachverhalt eingehen.**



Beispiele

- **Einfügen (Datenbank siehe Folie 49)**
 - insert into Maschinen values (77, "Bohrer")
 - insert into Maschinen values (93, "Sieb")
 - insert into Maschinen values (88,"Säge")

 - insert into PMZuteilung values (777,84,3)
 - insert into PMZuteilung values (82,93,2)





Löschen von Tupeln aus einer Relation

■ Einfachste Form beim Löschen

delete from <Relationen-Name>
where <Bedingung>

- Dabei werden alle Datensätze aus der Relation gelöscht, die die Bedingung erfüllen.
 - Soll genau ein Tupel gelöscht werden, wird die Bedingung aus einem Gleichheitstest mit dem Primärschlüssel bestehen.
- Beispiele
 - delete from Maschinen where mnr = 101
 - delete from PMZuteilung where note < 4



Sicherstellung der Integritätsbedingungen

- **Nach dem Löschen wird i. A. überprüft, ob die Integritätsbedingung noch erfüllt sind.**
 - Wenn nicht, wird die Operation rückgängig gemacht und ein Fehler gemeldet.
- **Art der Überprüfungen**
 - Primärschlüssel
 - Testen, ob ein Fremdschlüssel mit dem gleichen Wert noch in einer anderen Relation existiert.
- **Wir werden im Kapitel über Transaktionen noch genauer auf diesen Sachverhalt eingehen.**



3.2.1 Anfragen in SQL

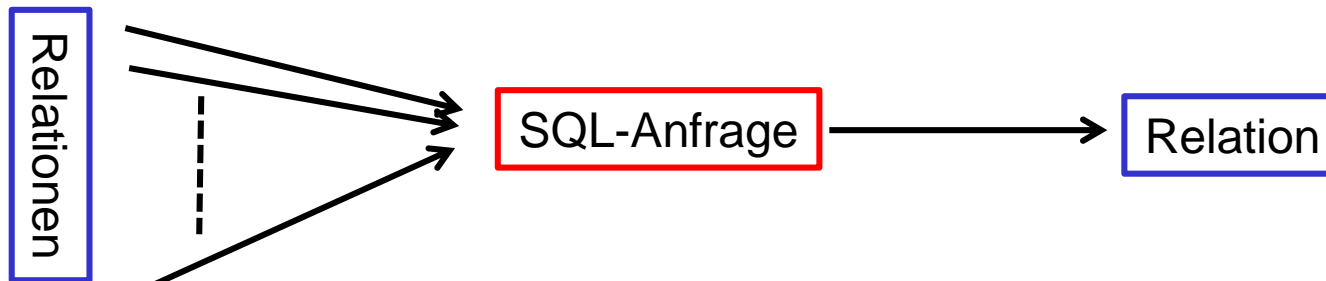
■ Grundschemata:

select < *Liste von Attributnamen* >
from < Liste von Relationennamen >
[where < Bedingung >]

```
select pnr  
from personal  
where abtnr = 10;
```

■ SQL-Anfrage

- Input: eine oder mehrere Relationen
- Output: eine temporäre Relation





3.2.2 Operatoren der relationalen Algebra in SQL

■ Ausgabe einer Relation der Datenbank

select *

from Maschine

- Bei Angabe von "*" in der select-Klausel werden alle Attribute der Relation aus der from-Klausel ausgegeben.

Ausgabefeld

	mnr	mname
	integer	character varying(10)
1	84	Presse
2	93	Füllanlage
3	101	Säge



Projektion

select distinct pnr
from PMZuteilung

Ausgabefeld

Datenanzeige Zerlegung

	pnr integer
1	73
2	82
3	333
4	114
5	67
6	51
7	69
8	701

select pnr
from PMZuteilung

Ausgabefeld

Datenanzeige Zerlegung

	pnr integer
1	67
2	67
3	67
4	73
5	114
6	114
7	51
8	69
9	333
10	701
11	701
12	82

- Die select-Klausel entspricht der Projektion in der relationalen Algebra (und nicht der Selektion).
- Ohne das Schlüsselwort distinct können Duplikate entstehen. Das Ergebnis wäre dann eine M-Relation.



Selektion (1)

■ Selektion

```
select *  
from PMZuteilung  
where Note < 4;
```

Ausgabefeld

Datenanzeige Zerlegung ▾

	pnr integer	mnr integer	note integer
1	67	84	3
2	67	93	2
3	67	101	3
4	114	101	3
5	51	93	2
6	69	101	2
7	333	84	3
8	701	84	2
9	701	101	2
10	82	101	2

■ Zugriff auf die Attribute

- Solange der Attributname eindeutig ist, kann direkt der Name in der where-Klausel verwendet werden.
- Ansonsten muss der Relationenname mit „.“ als Präfix verwendet werden.
- Atomare Ausdrücke mit Operatoren: =, >, <, !=, <=, >=, ...



Selektion (2)

■ Zusammengesetzte Bedingungen

■ and

select *
from PMZuteilung
where Note < 5 and Note > 2;

■ or

select *
from PMZuteilung
where Note < 5 or mnr = 84;

■ not

select *
from PMZuteilung
where not Note < 5 and mnr = 84;

Ausgabefeld

Datenanzeige Zerlegung ▾

	pnr integer	mnr integer	note integer
1	67	84	3
2	67	101	3
3	114	101	3
4	333	84	3

Output pane [Query 1]

Datenanzeige Zerlegung ▾

	pnr integer	mnr integer	note integer
1	67	84	3
2	67	93	2
3	67	101	3
4	73	84	5
5	114	101	3
6	51	93	2
7	69	101	2
8	333	84	3
9	701	84	2
10	701	101	2
11	82	101	2

Output pane [Query 1]

Datenanzeige Zerlegung ▾

	pnr integer	mnr integer	note integer
1	73	84	5



Selektion (3)

- **Aufbau komplexerer Bedingungen**
 - Verwendung von Quantoren
 - exists, any, some, all
 - Mengenoperatoren
 - in, not in
 - Spezialoperatoren
 - like (für Zeichenketten)

- **Details dazu gibt es später!**



Vereinigung

```
select pnr  
from PMZuteilung  
where Note < 4  
      union  
select pnr  
from Personal  
where abtnr = 10
```

Output pane [Query 1]

Datenanzeige		Zerlegung
	pnr integer	
1	82	
2	73	
3	333	
4	701	
5	114	
6	67	
7	51	
8	69	

- **Vereinigung union unterstützt Mengensemantik**
 - Duplikate werden eliminiert.
- **Schemaverträglichkeit ist wichtig.**
 - Gleichheit der Schemata wird jedoch in SQL nicht verlangt.



Differenz

```
select pnr from PMZuteilung  
      except  
select pnr from Personal where abtnr = 10;
```

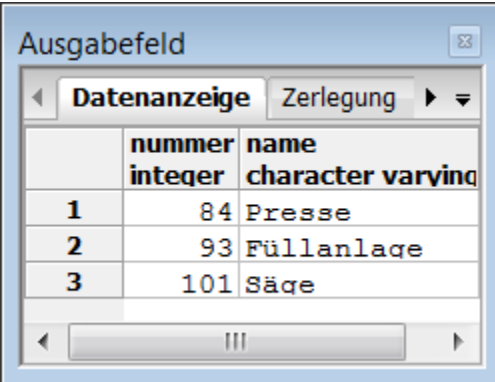
Datenanzeige		Zerleg
	pnr integer	
1	82	
2	333	
3	114	
4	51	
5	69	
6	701	

- Entsprechend gelten bei except die Regeln der Mengensemantik.
- Voraussetzung: Schemaverträglichkeit



Umbenennung

```
select mnr as nummer, mname as name  
from Maschinen;
```



	nummer integer	name character varying
1	84	Presse
2	93	Füllanlage
3	101	Säge

- Dadurch werden nur die Attribute der Relation Maschinen umbenannt.



Kartesisches Produkt

■ Kartesisches Produkt

select *

from Abteilung, Personal;

Ausgabefeld

	abtnr integer	aname character varying(10)	pnr integer	pname character varying(10)	vorname character varying(10)	abtnr integer	lohn character(2)
1	10	Spielzeug	67	Meier	Helmut	10	L4
2	10	Spielzeug	73	Müller	Margot	10	L5
3	10	Spielzeug	114	Bayer	Martin	63	L6
4	10	Spielzeug	51	Daum	Birgit	64	L7
5	10	Spielzeug	69	Störmer	Willi	64	L6
6	10	Spielzeug	333	Haar	Hans	63	L6
7	10	Spielzeug	701	Reiner	Willi	64	L6
8	10	Spielzeug	82	Just	Michael	64	L6
9	63	Computer	67	Meier	Helmut	10	L4
10	63	Computer	73	Müller	Margot	10	L5

...

...

...

...

- Attribute der Schemata müssen hierbei **nicht** disjunkt sein.
- Ein Join kann durch eine zusätzliche where-Klausel formuliert werden.
 - where Abteilung.abtnr = Personal.abtnr



Reihenfolge bei der Auswertung einfacher SQL-Anfragen

- Einfache Anfragen sind die SQL-Anfragen, die aus einer **Select**-, **From**- und **Where**-Klausel bestehen.
 - Andere Klauseln werden wir noch später behandeln!
- **Logische Umsetzung der Anfrage in folgender Reihenfolge:**
 1. Auswertung der from-Klausel: Kartesisches Produkt
 2. Auswertung der where-Klausel: Filtern der Tupel
 3. Auswertung der select-Klausel: Projektion der Attribute
- **Beispiel**
 - Gesucht sind die Personalnummern der Angestellten aus mit Lohn von 60000 und höher.



Multimengensemantik

■ Projektion

- Ohne distinct keine Duplikatbeseitigung.
 - select pnr from PMZuteilung;

■ Vereinigung

- union all
select pnr from PMZuteilung where note < 3
union all
select pnr from Personal where Lohn > 65000;
- Alle Elemente beider Relationen bleiben erhalten!

■ Differenz

- except all



Beispiele (1)

- Welche Angestellten verdienen 65000 oder 51000?



Beispiele (2)

- Gesucht sind die Gehälter der Angestellten, die Maschine 84 bedienen können.



Beispiele (3)

- Gesucht sind die Personalnummern der Angestellten, die Maschine 84 bedienen können und mindestens 60000 verdienen.



Beispiele (4)

- Gesucht sind die Personalnummern der Angestellten, die Maschine 84 bedienen können, aber nicht mehr als 60000 verdienen.



3.2.3 where-Klausel ohne Unteranfragen im Detail

■ Wiederholung

- Formeln bestehen aus Atomen der Form
$$“A \theta B”$$
 - θ steht für ein Vergleichsoperator
 - A und B stehen hier nicht nur für Attribute und Konstanten, sondern es können auch komplexere Ausdrücke sein.
 - **Numerische** Attribute: Nutzung der arithmetischen Grundoperatoren und zusätzlicher Funktionen (z. B. abs).
 - **Zeichenketten**: Verkettung “||”.
- Atome können mit den Operatoren not, or und and zu komplexeren Formeln verknüpft werden.



Atomare Formeln

■ **between-Operator**

- A between B and C
- Ausdruck ist äquivalent zu $B \leq A \text{ and } A \leq C$

■ **like-Operator**

- A like B
- Verwendung bei Zeichenketten, um einfache Mustersuchen zu unterstützen.
 - Test auf Gleichheit von Zeichenketten, wobei Wildcards benutzt werden können.
 - % repräsentiert beliebig viele Zeichen
 - _ repräsentiert genau ein Zeichen
- Beispiel:
select PName
from Personal
where Vorname like 'M%g_t'



Atomare Formeln (2)

■ in-Operator

- Es gibt zwei Varianten des in-Operators. Hier zunächst mal die einfachere.

$A \text{ in } (b, c, \dots, z)$

- Dabei ist A ein Ausdruck und b, \dots, z Konstanten.
- Dieser Ausdruck ist äquivalent zu
 - $A = b \text{ or } A = c \text{ or } \dots \text{ or } A = z$



Sonderbehandlung für NULL-Werte

■ Die beiden Anfragen

- select * from PMZuteilung where note > 3;
- select * from PMZuteilung where note ≤ 3;

Das ist doch das
negierte Prädikat!!

liefern den Datensatz (82,84,null) nicht zurück!

→ Der Datensatz kann nicht wie üblich gefunden werden.

■ Lösung

- Es kann explizit auf den Wert null getestet werden.
- Beispiel
 - select * from PMZuteilung where note is null;

Ausgabefeld

	pnr integer	mnr integer	note integer
1	82	84	



... und damit hat man eine **dreiwertige Logik!**

- Die mit null-Werten aufgefüllten Attribute erfordern eine dreiwertige Logik
 - Zusätzlich zu dem Wert **true** und **false** kann eine Bedingung den Wert **unknown** liefern.
- Das Ergebnis einer atomaren Formel ist
 - **unknown**, wenn ein null-Wert mit einem anderen Wert durch einen relationalen Operator verglichen wird.
- Bei zusammengesetzten Formeln wird eine Erweiterung der zweiwertigen Logik benötigt.



Die Wahrheitstabellen der dreiwertigen Logik

AND	true	false	unknown
true	true	false	?
false	false	false	?
unknown	?	?	?

OR	true	false	unknown
true	true	true	?
false	true	false	?
unknown	?	?	?

NOT	true	false	unknown
	false	true	?

Eine Selektion (where-Klausel) liefert nur die Datensätze, die bei der Auswertung den Wert true liefern.



3.2.4 Joins

- Joins werden in SQL durch kartesisches Produkt und Selektion ausgedrückt.

- Equi-Join der Relationen r und s

select * from **r, s** where **r.A = s.B**

Join-Bedingung

Relationen

- Hiermit lassen sich auch Semi-Joins ausdrücken

- select **r.*** from r, s where r.A = s.B

Liefert alle Attribute der Relation r

- Das Datenbanksystem nutzt clevere Strategien, um die Berechnung des kartesischen Produkts zu vermeiden.



Joins in der from-Klausel

- Joins können direkt in der from-Klausel formuliert werden.
 - „Innere“ Join-Varianten
 - Das Schlüsselwort inner kann weggelassen werden.
 - Innerer Theta-Join
 - from r inner join s on r.A > s.B
 - Innerer Equi-Join über ein gemeinsames Attribut A
 - from r inner join s using (A)
 - Resultatschema enthält das Attribut A genau einmal.
 - Natural Join
 - from R natural inner join S



Outer-Joins

- **Outer-Joins sind spezielle Joins**
 - Ergebnis umfasst alle Tupel des äquivalenten inneren Join
 - **Zusätzlich** werden noch die Tupel, die keinen Join-Partner haben, in das Ergebnis aufgenommen.
 - Die fehlenden Werte werden mit dem Wert **null** aufgefüllt.
- **Folgende Outer-Joins existieren**
 - left outer join
 - from r left outer join s on r.A = s.B
 - right outer join
 - from r right outer join s on r.A = s.B
 - full outer join
 - from r full outer join s on r.A = s.B



Beispiel

 r_1

A	B
a1	b1
a2	b2

 r_2

B	C
b1	c1
b3	c2

r_1 left outer join r_2

A	B	C
a1	b1	c1
a2	b2	null

r_1 right outer join r_2

A	B	C
a1	b1	c1
null	b3	c2

r_1 full outer join r_2

A	B	C
a1	b1	c1
a2	b2	null
null	b3	c2



Joins über gleiche Relationen

- Werden bei einem Join gleiche Relationen benutzt, muss mindestens eine davon über eine **Tupelvariable** angesprochen werden.

- Das Konzept der Tupelvariable kennen wir bereits vom Tupelkalkül.

- **Beispiel**

- Welche Angestellten können die gleiche Maschine bedienen?

```
select distinct a1.pnr, a2.pnr  
from PMZuteilung a1, PMZuteilung a2  
where a1.mnr = a2.mnr and a1.pnr < a2.pnr
```

- Dabei sind a1 und a2 Tupelvariablen.
- Die Deklaration der Tupelvariablen erfolgt in der from-Klausel.



3.2.5 select-Klausel

■ Ausgabe aller Attribute

- select * from r, s, ... where

Alle Attribute der Relationen r, s,... werden ausgegeben.

■ Ausgabe aller Attribute einer Relation

- select r.* from r,s... where ...

Alle Attribute der Relation r werden ausgegeben.

- Beispiel

- select Personal.* from Personal, PMZuteilung using (pnnr)

■ Wiederholung: Mengen- und Multimengensemantik

- Mengensemantik: Beseitigung der Duplikate

- select distinct pnr from PMZuteilung

- Multimengensemantik: Erhaltung der Duplikate

- select pnr from PMZuteilung



Umbenennung und Berechnung

- **Umbenennung von Attributen**
 - select mnr as m, pnr as p from PMZuteilung;
- **Verwendung der map-Operation**
 - **Ausdrücke** und **Funktionen** (Ann.: $RS_r = \{B, C, D\}$)
 - select **B*C** as Y, **abs**(D) as Z from r
 - Die select-Klausel verhält sich wie ein Map-Operator
 - Die Anzahl der Tupel in der Eingaberelement bleibt gleich!
 - Beispiel
 - Vorlesung
- **Behandlung von **Unterabfragen** in der select-Klausel → später!**



3.2.6 Skalare Aggregate

- **Skalare Aggregatfunktion**
 - Liefert zu einer Menge/Multimenge von Werten einen Wert zurück.

- **Anwendung von Aggregatfunktionen in SQL**
 - Die Aggregatfunktion wird auf eine Spalte in einer Tabelle angewendet.
 - select count(pnr) from PMZuteilung;

 - *Spalten können durch einen Ausdruck oder eine Funktion dynamisch erzeugt werden.*
 - select count(Note/2) from PMZuteilung;



Aggregatfunktionen in SQL

■ Typische numerische (unäre) Aggregate

- count, sum, avg, min und max.

■ Weitere statistische Aggregate

- variance, corr, stddev, regr_slope

■ Logische Aggregate (liefern true oder false)

- exists, every, any, some

→ Diese werden später besprochen.



Aggregate min und max

- **Aggregate min und max**
 - Liefern zu einem mengenwertigen Ausdruck, das kleinste bzw. das größte Element der Menge.
 - Null-Werte werden dabei ignoriert.
- **Beispiel**
 - Liefere für die Angestellten der Abteilung A4 die beste und schlechteste Note bei der Bedienung einer Maschine.

```
select min(Note), max(Note)  
from ...
```



Aggregatfunktion count

■ Zwei Varianten

- count(*) - Anzahl der Tupel in einer Menge.

- Beispiel:

select count(*) from PMZuteilung where mnr = 93;

- count(A) – A ist ein Attribut oder ein Ausdruck

- Ergebnis ist dann die Anzahl der Werte in der Menge (ohne die NULL-Werte)

- Falls die Menge leer oder nur aus Nulleverten besteht, wird die **Zahl 0** geliefert.

- Beispiel

- select count(A) from r;

r	A	B
	a1	2
	a1	5
	NULL	2



Aggregatfunktionen sum, avg

- **sum**

- Berechnung der Summe über einen multimengenwertigen Ausdruck.

- **avg**

- Berechnung des Durchschnitts über einen mengenwertigen Ausdruck.

- **Beispiel**

- select sum(B), avg(B) from r;

- **Unterschied von sum und avg zu count**

- Bei einer leeren Eingabe wird der Wert NULL und nicht die Zahl 0 zurückgeliefert.



Aggregate mit distinct

- Bei Eingabe einer Multimenge kann mit distinct vor dem Aggregat zunächst eine Duplikatelimininierung vorgenommen werden.
 - Beispiele
 - select count(distinct A), avg(distinct B) from r;
- In count(*) kann distinct nicht genutzt werden.



3.2.7 Vektoraggregate

■ Motivation

- Berechnung von Aggregaten liefert genau einen Wert.
- Beispiel:
 - Berechne die Anzahl der Angestellten, die Maschine 42 bedienen können.
 - Wunsch: Berechne dieses Aggregat für alle Maschinen
→ Formulierung der Anfrage für jede Maschine?

■ Stattdessen

- Verwendung einer Anfrage mit einer group-by-Klausel
 - Siehe Operator γ der erweiterten RA



Gruppierung in SQL

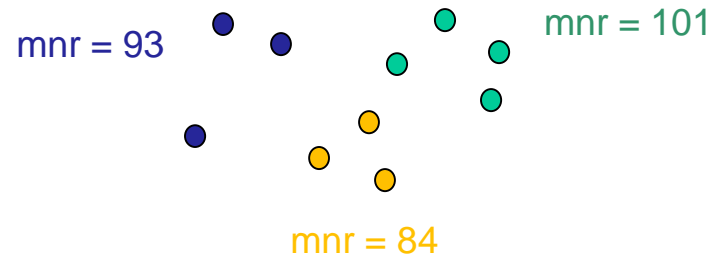
- **Für die Gruppierung gibt es eine Group-by-Klausel**
 - Diese Klausel beginnt mit dem Schlüsselwort group by
 - und steht direkt hinter der optionalen where-Klausel
 - Die Group-by-Klausel besteht aus
 - einer **Liste von Attributen**
 - Relation wird partitioniert in Äquivalenzklassen bzgl. der Gleichheit in den angegebenen Attributen.
 - Diese Attribute müssen auch in der select-Klausel stehen.
 - Aber sie müssen nicht notwendigerweise im Schema einer der Relationen in der from-Klausel sein.
 - **Aggregate in der select-Klausel** werden für jede Äquivalenzklasse berechnet.
 - Keine Aggregate vorhanden → Duplikateliminierung



Beispiel

■ Beispiel

- select **mnr**, count(*)
from PMZuteilung
group by **mnr**;



■ Gruppierung mit mehreren Attributen

select pnr, note, count(*)
from PMZuteilung
group by pnr, note;

■ Gruppierung mit einem Ausdruck

select **note/2 as n**, count(*)
from PMZuteilung
group by **n**;



Null-Werte

- Gibt es einen Null-Wert in der Spalte, wird eine eigene Gruppe erzeugt.
 - Das Aggregat wird wie üblich berechnet.
- Mehrere Gruppierungsattribute mit Null-Werten
 - NULL steht in jedem Attribut für einen Wert
 - Beim Gruppieren wird dies entsprechend umgesetzt.

tmp3

	x integer	y integer	z integer
1		1	2
2		2	3
3	1		2
4	2		3
5	1	1	2
6	1	1	3



```
select x,y,count(*)  
from tmp3  
group by x,y;
```



Output pane [Query 1]

	x integer	y integer	count bigint
1		1	1
2	1		1
3	1	1	2
4	2		1
5		2	1



Having

■ Motivation

- Bei der Berechnung von Durchschnittsnoten könnten nur die Äquivalenzklassen interessant, die genügend **viele** Datensätze haben.

→ siehe z. B. Vorlesungsevaluation der Fachschaft

■ Having-Klausel

- Filtern von Gruppen, die gewisse Bedingungen erfüllen.
 - wie z. B. Anzahl der Tupel in einer Gruppe > 5 .
- Die Having-Klausel steht direkt hinter der Group-By-Klausel und beginnt mit dem Schlüsselwort having.
 - Danach folgt ein Prädikat, dass für eine Gruppe genau einen Booleschen Wert (true oder false) liefert.
- Nur die Gruppen, für die das Prädikat true liefert, werden in der Anfrage berücksichtigt.



Prädikate in der Having-Klausel

- **Einschränkung der Prädikate**
 - Attribute aus der Group-By-Klausel
 - Aggregatfunktionen
 - Liefern pro Gruppe genau einen Wert

- **Beispiel**
 - select **mnr**, **avg**(note)
from PMZuteilung
group by **mnr**
having **count**(*) > 2;



3.2.8 Sortierte Ausgabe

■ Motivation

- Ausgabe der Angestellten sortiert bzgl. der Durchschnittsnote in PMZuteilung.

■ Order-Klausel

- beginnt mit dem Schlüsselwort order by
 - direkt hinter der optionalen Having-Klausel
- Danach folgt eine Liste von Sortierkriterien
 - Sortierung wird bestimmt durch erstes Kriterium. Im Fall von Gleichheit wird das zweite Kriterium genutzt, usw.

```
ORDER BY sort_expression1
        [ASC | DESC] [NULLS { FIRST | LAST }]
        [, sort_expression2
        [ASC | DESC] [NULLS { FIRST | LAST }] ...]
```



Sortierkriterien

■ Attribute

- Entweder im Schema der Relationen aus der from-Klausel vorhanden.
- oder die erst in der Select-Klausel definiert wurden.

■ ASC|DESC

- Option, ob aufsteigend oder abtsteigend sortiert wird.
 - Default: Aufsteigend (ASC)

■ NULLS { FIRST | LAST }

- Behandlung von Nullwerten
 - zuerst oder am Ende der sortierten Ausgabe
- Erst seit SQL:2003 im Standard



Einschränkung der Resultate

■ Motivation

- Man möchte die drei besten Angestellten (beste Durchschnittsnote) auszeichnen.
 - Sortieren der Daten und auslesen der ersten drei Tupel.

■ Limit-Klausel

■ limit N [offset M]

- N, M sind ganze Zahlen
- Anfrage liefert aus der sortierten Ergebnisfolge das (M+1)-te, (M+2)-te, ... (M+N-1)-te Tupel.

- Sinnvoll nur bei SQL-Anfragen mit order-by-Klausel.

→ Optimierungspotential bei der Auswertung der Anfrage!

- Syntax noch nicht im SQL-Standard, aber wird bereits von vielen Datenbanksystemen unterstützt.



Beispiele

- **Sortiere die Tabelle PMZuteilung absteigend bzgl. der Note.**
 - select * from PMZuteilung
order by Note desc;
- **Sortiere die Tabelle PMZuteilung bzgl. der Note und dann bzgl. der Personalnummer.**
 - select * from PMZuteilung
order by Note desc, pnr;
- **Liefere die ersten fünf Ergebnisse der vorhergehenden Anfrage, aber NULL-Werte hinten.**
 - select * from PMZuteilung
order by Note desc, pnr nulls last
limit 5;



Beispielanfragen

- Wie viele Angestellte können Maschine 93 bedienen?
- Liefere für jede Maschine die Anzahl der Angestellten, die diese Maschine bedienen können.
- Liefere für jede Maschine die Anzahl der Angestellten, die diese Maschine bedienen können, aber nur die Maschinen mit höchstens 4 Angestellten.
- Sortiere PMZuteilung nach dem Attribut Note und dann nach dem Namen des Angestellten.
- Liefere nur die ersten drei Antworten der letzten Anfrage.



Zusammenfassung

■ Zusammensetzung einer SQL-Anfrage

select X
from R,S,T,...
where F
group by Y
having G
order by H
limit N offset M

← noch nicht im Standard !!

- X eine Menge von Attributen
- R,S,T,... eine Liste von Relationen
 - Optional können bereits hier die Joins formuliert werden.
- F eine Boolesche Formel
- Y eine Menge von Attributen
- G eine Boolesche Formel zur Filterung von Gruppen
- H eine Liste von Attributen zum Sortieren