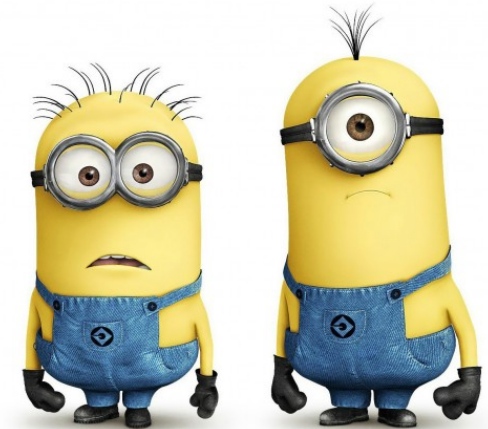


Datenbanken SQL

Basierend auf den Folien S. Wagner

Lernziele

- Wie arbeite ich mit SQL?
 - Tabellen anlegen
 - Tabellen füllen
 - Anfragen stellen



Reminder: relationales Datenbankmodell I

Seien D_1, D_2, \dots, D_n Domänen (\sim Wertebereiche)

- Relation: $R \subseteq D_1 \times \dots \times D_n$

Bsp.: Telefonbuch $\subseteq \text{string} \times \text{string} \times \text{integer}$

- Tupel: $t \in R$

Bsp.: $t = (\text{„Marty McFly“}, \text{„Main Street“}, 4711)$

- Schema: legt die Struktur der gespeicherten Daten fest

Bsp.:

Telefonbuch: $\{[\text{Name: string, Adresse: string, } \underline{\text{Telefon\#:integer}}]\}$

Reminder: relationales Datenbankmodell II

Name	Strasse	<u>Tel.Nummer</u>
Marty McFly	Main Street	4711
Herman Monster	Mockingbird Lane	0666
James Bond	Kings-Cross	6007

- Ausprägung: der aktuelle Zustand der Datenbasis
- Schlüssel: minimale Menge von Attributen, deren Werte ein Tupel eindeutig identifizieren
- Einer der Schlüsselkandidaten wird als Primärschlüssel ausgewählt
- Primärschlüssel wird unterstrichen
- Der Primärschlüssel hat eine besondere Bedeutung bei der Referenzierung von Tupeln

SQL Historie

- Erfunden 1970 von Donald D. Chamberlin and Raymond F. Boyce bei IBM
- Damals noch SEQUEL (*Structured English Query Language*) genannt
- Wegen Markenrechtsproblemen umbenannt in SQL (*Structured Query Language*)
- Zunächst exklusive für System R
- Ab 1979 auch für Oracel, VAX,...
- Seit 1986 standardisiert durch ANSI

- Standardisierte
 - Datendefinitions (DDL)- Sprache
 - Datenmanipulations (DML)- Sprache
 - Anfrage (Query)-Sprache
- Derzeit aktueller Standard ist SQL 99 und SQL 11 (2011)
 - objektrelationale Erweiterung
 - Unterstützung für temporale Datenbasen
- Für praktische Übungen steht eine Web-Seite zur Verfügung:

<http://www-db.in.tum.de/research/publications/books/DBMSeinf>

SQL Datentypen

- character (n), char (n)
- character varying (n), varchar (n)
- bit (n)
- bit varying (n)
- numeric (p,s) bzw. Decimal (p,s), integer, float, real
- Blob (binary large object) oder raw für sehr große binäre Daten
- clob (character large object) für sehr große String-Attribute
- date für Datumsangaben
- xml für XML-Dokumente

Anlegen von Tabellen

- CREATE TABLE BspTabelle
(wert1 INTEGER,

Name	Strasse	<u>Tel.Nummer</u>
Marty McFly	Main Street	4711
Herman Munster	Mockingbird Lane	0666
James Bond	Kings-Cross	6007

t',

uniqueWert5 INTEGER UNIQUE,
keyWert6 INTEGER PRIMARY KEY);

- CREATE TABLE telefonbuch
(Name VARCHAR(256), Strasse VARCHAR(256), Tel.Nummer
INTEGER PRIMARY KEY);

Anlegen von Tabellen

- `CREATE TABLE BspTabelle`
(wert1 INTEGER,
defaultWert2 VARCHAR(255) DEFAULT `Unbekannt`,
mayBeNullWert3 VARCHAR(255) NULL,
notNullWert4 VARCHAR(255) NOT NULL,
uniqueWert5 INTEGER UNIQUE,
keyWert6 INTEGER PRIMARY KEY);
- `CREATE TABLE telefonbuch`
(Name VARCHAR(256), Strasse VARCHAR(256), Tel.Nummer
INTEGER PRIMARY KEY) VALUES (Marty McFly, Main Street,
4711);

Daten einfügen

- INSERT INTO BspTabelle

Name	Strasse	<u>Tel.Nummer</u>
Marty McFly	Main Street	4711
Herman Munster	Mockingbird Lane	0666
James Bond	Kings-Cross	6007

- INSERT INTO telefonbuch
(Name, Strasse, Tel.Nummer)
VALUES
(John Doe, Nowhere Road, 0181);

Daten einfügen

- INSERT INTO BspTabelle
(Spaltenname1,...SpaltennameN)
VALUES
(wert1,...,wertN);
- INSERT INTO telefonbuch
(Name, Strasse, Tel.Nummer)
VALUES
(John Doe, Nowhere Road, 0181);

Verändern und löschen von Daten

- UPDATE BspTabelle

SET Spaltenname1 = neuerWert

Ohne WHERE:
Verändert alle
Einträge

Name	Strasse	<u>Tel.Nummer</u>
Marty McFly	Main Street	4711
Herman Munster	Mockingbird Lane	0666
James Bond	Kings-Cross	6007

WHERE Spaltenname = zuloschenderWert;

Ohne WHERE: Löscht
alle Einträge in der
Tabelle

- UPDATE telefonbuch

SET Name = John Smith

WHERE Name = James Bond;

- DELETE FROM telefonbuch

WHERE Tel.Nummer = 4711;

Verändern und löschen von Daten

- UPDATE BspTabelle
SET Spaltenname1 = neuerWert
WHERE Spaltenname2 = wert;

Ohne WHERE:
Verändert alle
Einträge

- DELETE FROM BspTabelle
WHERE Spaltenname = zulöschenderWert;

Ohne WHERE: Löscht
alle Einträge in der
Tabele

- UPDATE telefonbuch
SET Name = John Smith
WHERE Name = James Bond;
- DELETE FROM telefonbuch
WHERE Tel.Nummer = 4711;

SQL Abfragen I

- `SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle`
- `SELECT * FROM BspTabelle`
- `SELECT (DISTINCT) Spaltenname1,...,SpaltennameN FROM BspTabelle`
`WHERE Spaltenname1 Operator1 zuTestenderWert1`
`AND / OR Spaltenname2 Operator2 zuTestenderWert2`
`ORDER BY Spaltenname DESC / ASC`

SQL Abfragen I

- `SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle`
- `SELECT * FROM BspTabelle`
- `SELECT (DISTINCT) Spaltenname1,...,SpaltennameN FROM BspTabelle`
`WHERE Spaltenname1 Operator1 zuTestenderWert1`
`AND / OR Spaltenname2 Operator2 zuTestenderWert2`
`ORDER BY Spaltenname DESC / ASC`

SQL Abfragen II

Operator	Beschreibung
=	Gleichheit
<> (manche Dialekte auch !=)	Ungleichheit
>	Größer als
<	Kleiner als
>=	Größer oder Gleich
<=	Kleiner oder Gleich
BETWEEN	Innerhalb eines Bereichs (oft unterschiedlich interpretiert von DB zu DB)
LIKE	Suche nach Mustern mit % und _ als Wildcards
IN	Ergebnisraum beschränken auf bestimmte Werte

SQL Abfragen III

Name	Strasse	<u>Tel.Nummer</u>
Marty McFly	Main Street	4711
Herman Munster	Mockingbird Lane	0666
James Bond	Kings-Cross	6007

- `SELECT Name FROM Telefonbuch WHERE Name LIKE 'H%';`
- `SELECT Name FROM Telefonbuch WHERE Tel.Nummer LIKE '_666';`
- `SELECT Strasse FROM Telefonbuch WHERE Tel.Nummer BETWEEN '1000' AND '5000';`
- `SELECT Name FROM Telefonbuch WHERE Strasse IN ('Mockingbird Lane', 'Broad Way');`

SQL Abfragen über mehrere Tabellen

Personal		
<u>PersonalNR.</u>	Name	AbteilungsNr.
27004	A. Einstein	22
38002	H. Ford	25
41573	A. Smith	20

Abteilungen		
<u>AbteilungsNR.</u>	Bezeichnung	Abteilungsleiter
20	Controlling	41573
21	Marketing	69547
22	R&D	69004
25	Produktion	38002

- `SELECT p1.Name, a1.Bezeichnung FROM Personal p1, Abteilung a1 WHERE p1.PersonalNR. = a1.Abteilungsleiter`

„Do it yourself“ SQL

Aufgabe:

Die Firma Arcade-Paradies hat eine Datenbank. Die Datenbank enthält die Details zu den zum Verkauf stehenden Spielautomaten, der im Lager verfügbaren Automaten und eine Liste der Bestellungen. Die Bestellnummer setzt sich hierbei aus einem Buchstaben (I für Inlandsbestellungen oder A für Auslandsbestellungen), einer laufenden Nummer und dem Datum der Bestellung zusammen. Formulieren Sie die folgenden SQL-Befehle.

Form: Murmelgruppe

Zeit: 10 min



„Do it yourself“ SQL

Details

<u>Art. Nummer</u>	Name	Erscheinungsjahr	Genre
00324124	Rick Dangerous	1989	Jump'n'Run
00537772	Double Dragon	1987	Beat'em Up
00635525	Asteroids	1979	Shoot'em Up

Lagerbestand

<u>Lagerkennung</u>	Art.Nummer	Anzahl	Preis
L001	00324124	15	29,95
L002	00537772	3	129,99
L003	00635525	0	529,79

Bestellungen

<u>Bestellnummer</u>	Art.Nummer	Anzahl	Adresse
I.0091.131112	00537772	1	Somewhere, Someone
A.0047.141112	00635525	1	Faraway, SomeStranger

„Do it yourself“ SQL

Geben Sie folgende SQL-Befehle an:

- Erstellen der Tabellen *Details* und *Bestellungen*
- Einfügen der Daten in die Tabelle *Lagerbestand*
- Einfügen des Spiels *X-Out*, erschienen *1989*, Genre *Shoot'em Up* in einer Anzahl von *5* Stück mit einem Preis von *13,37*.
- Löschen des Automaten *Asteroids* aus dem System
- Angabe aller Adressen von Bestellungen von Automaten mit dem Erscheinungsjahr *1989*
- Angabe der *Lagerkennung* aller *Shoot'em Up* Automaten

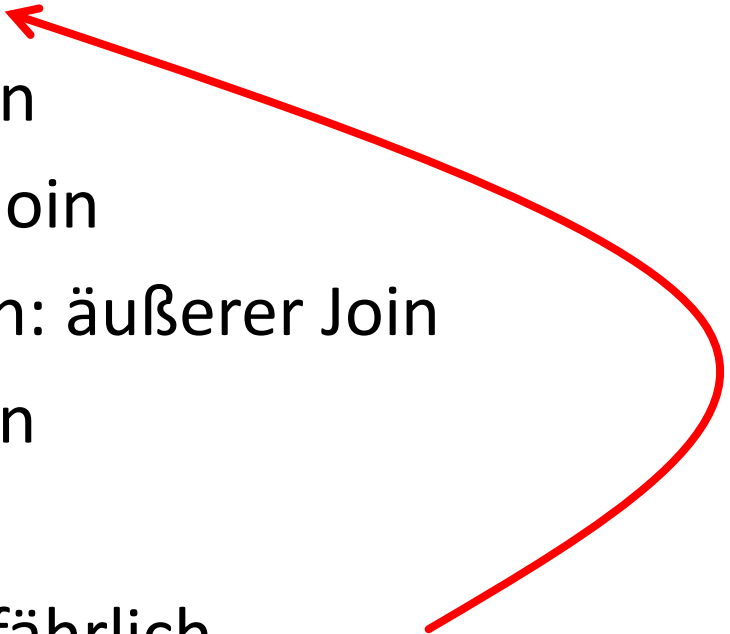


„Do it yourself“ SQL

- Erstellen der Tabellen *Details* und *Bestellungen*
CREATE TABLE Details (Art.Nummer int, Name varchar(256), Erscheinungsjahr date, Genre varchar(256))
- Einfügen der Daten in die Tabelle *Lagerbestand*
INSERT INTO Lagerbestand VALUES ('L001', 00324124, 15, 29,95)
- Löschen des Automaten *Asteroids* aus dem System
DELETE FROM Lagerbestand WHERE (SELECT Art.Nummer FROM Details WHERE Name='Asteroids')
DELETE FROM Details WHERE Name='Asteroids'
- Angabe der *Lagerkennung* aller *Shoot'em Up* Automaten
SELECT Lagerkennung FROM Lagerbestand WHERE Art.Nummer=(SELECT Art.Nummer FROM Details WHERE Genre='Shoot'em Up')



Joins

- cross join: Kreuzprodukt
 - natural join: natürlicher Join
 - Join oder inner join: Theta-Join
 - left, right oder full outer join: äußerer Join
 - union join: Vereinigungs-Join
-
- Sehr mächtig, aber auch gefährlich
 - <http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html>
- 

Nullwerte

- Unbekannter Wert, der vielleicht später nachgereicht wird
- Nullwerte können auch im Zuge der Abfrageauswertung entstehen (Bsp. äußere Joins)
- Manchmal entstehen sehr überraschende Abfrageergebnisse, wenn Nullwerte vorkommen
- **Beispiel**
select count (*)
from Studenten
where Semester < 13 or Semester > =13
- Wenn es Studenten gibt, deren Semester-Attribut den Wert null hat, werden diese nicht mitgezählt
- Der Grund liegt in folgenden Regeln für den Umgang mit **null**-Werten begründet:

Auswertung von Null-Werten I

1. In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis null. Dementsprechend wird z.B. **null** + 1 zu **null** ausgewertet, aber auch **null** * 0 wird zu **null** ausgewertet.
2. SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente **null** ist. Beispielsweise wertet SQL das Prädikat (PersNr=...) immer zu **unknown** aus, wenn die PersNr des betreffenden Tupels den Wert **null** hat.
3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Auswertung von Null-Werten II

not	
true	false
unknown	unknown
false	true

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

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

=		
true	true	true
true	false	false
true	unknown	unknown
false	false	true
false	unknown	unknown
unknown	unknown	unknown

Auswertung von Null-Werten III

- Diese Berechnungsvorschriften sind recht intuitiv.
Unknown or **true** wird z.B. zu **true** - die Disjunktion ist mit dem **true**-Wert des rechten Arguments immer erfüllt, unabhängig von der Belegung des linken Arguments. Analog ist **unknown and false** automatisch **false** - keine Belegung des linken Arguments könnte die Konjunktion mehr erfüllen.
- 4. In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** ausgewertet, nicht ins Ergebnis aufgenommen.
- 5. Bei einer Gruppierung wird **null** als ein eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

```
delete from Voraussetzungen
where Vorgänger in (select
Nachfolger
from Voraussetzungen);
```

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

```
delete from Voraussetzungen
where Vorgänger in (select
Nachfolger
from Voraussetzungen);
```

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

```
delete from Voraussetzungen
where Vorgänger in (select
Nachfolger
from Voraussetzungen);
```

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5043	5052
5052	5229

```
delete from Voraussetzungen
where Vorgänger in (select
Nachfolger
from Voraussetzungen);
```

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5043	5052
5052	5229

```
delete from Voraussetzungen
where Vorgänger in (select
Nachfolger
from Voraussetzungen);
```


Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5052	5229



Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

delete from Voraussetzungen
where Vorgänger in (**select**
 Nachfolger
from Voraussetzungen);

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

1

2

1

5

2-stufige Änderung mit Markierung

```
delete from Voraussetzungen  
where Vorgänger in (select  
Nachfolger  
from Voraussetzungen);
```

Abhängigkeiten bei Veränderungen

Voraussetzungen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049

2-stufige Änderung mit Markierung

Mögliche Markierung:

1. Postfix für Wert
z.B. Wert+del
2. Extra Spalte für
Veränderung „Dirty Bit“

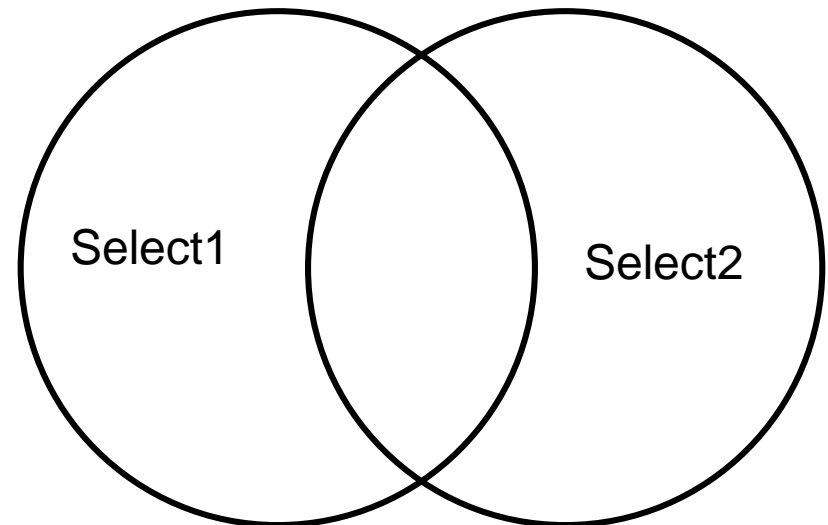
```
delete from Voraussetzungen  
where Vorgänger in (select  
Nachfolger  
from Voraussetzungen);
```

Mengenoperationen

- UNION, INTERSECT, MINUS
- SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle1

UNION / INTERSECT / MINUS

SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle2

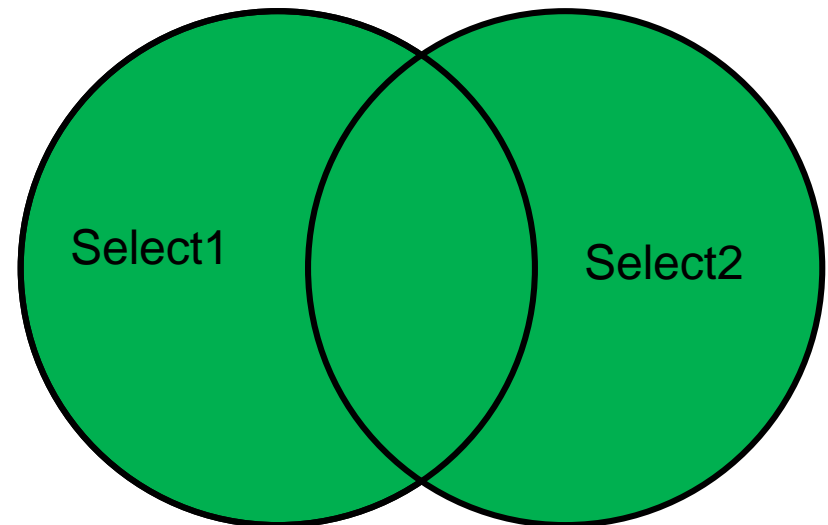


Mengenoperationen

- UNION, INTERSECT, MINUS
- SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle1

UNION / INTERSECT / MINUS

SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle2

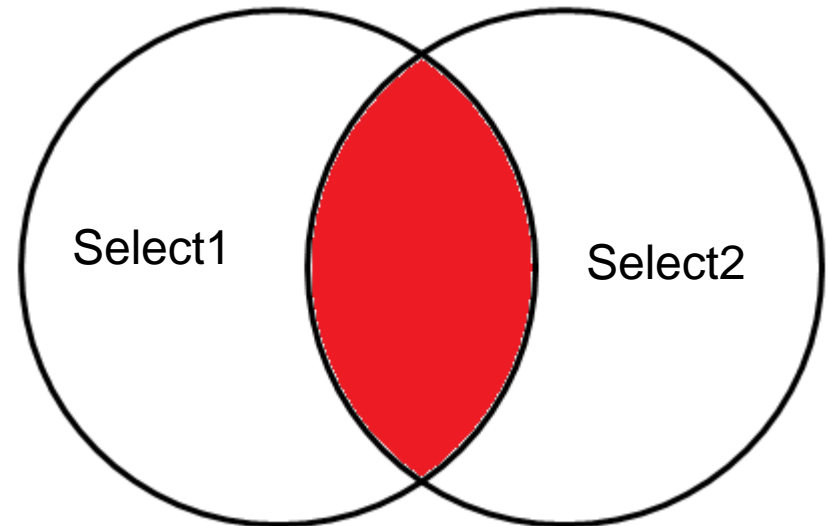


Mengenoperationen

- UNION, INTERSECT, MINUS
- SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle1

UNION / **INTERSECT** / MINUS

SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle2

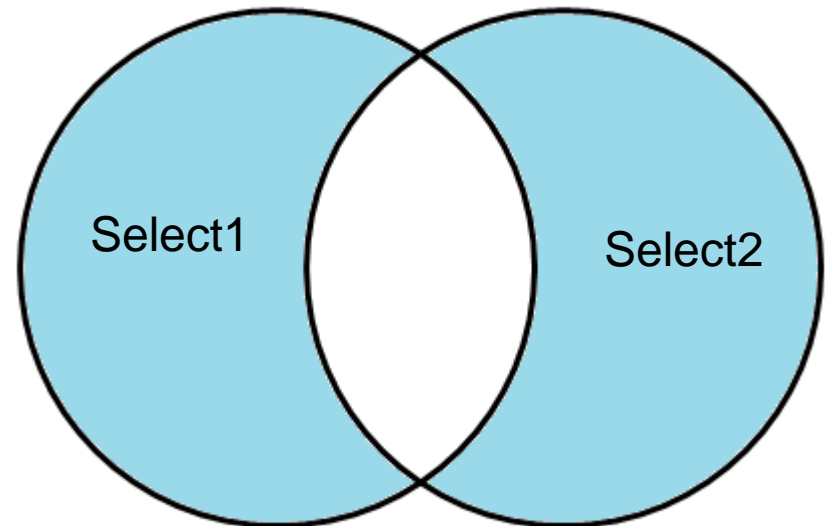


Mengenoperationen

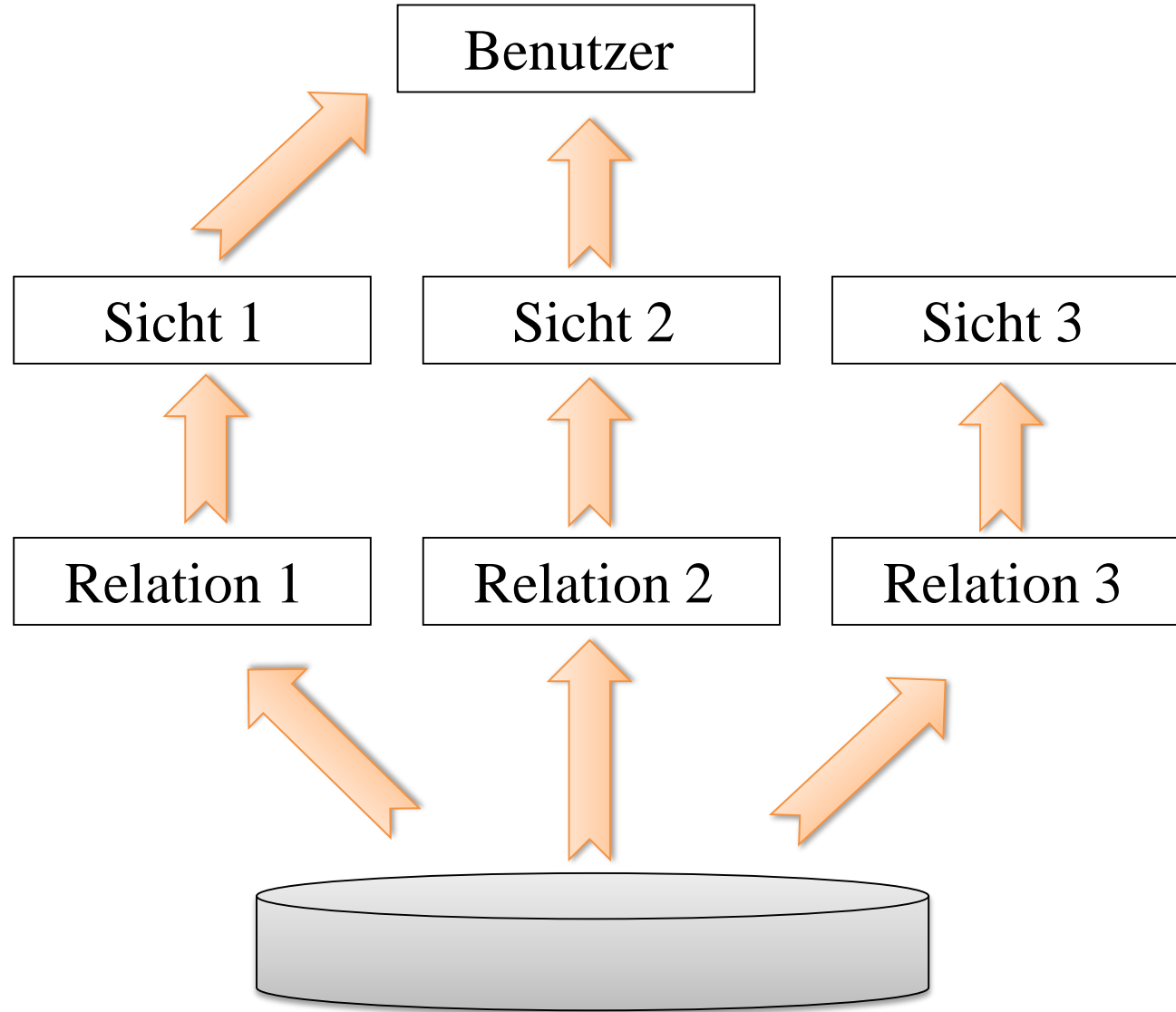
- UNION, INTERSECT, MINUS
- SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle1

UNION / INTERSECT / MINUS

SELECT Spaltenname1,...,SpaltennameN FROM BspTabelle2



Views I



Views II

- CREATE VIEW BspSicht AS SELECT
Spaltenname1,...,SpaltennameN FROM BspTabelle
- DROP VIEW BspSicht
- Views sind „virtuelle“ Tabellen
- Werden vom DBMS falls möglich aktuell gehalten
 - nur eine Basisrelation
 - Schlüssel muß vorhanden sein
 - keine Aggregatfunktionen, Gruppierung und Duplikateliminierung