

# 1. Relationale Algebra

## Grundlagen des relationalen Modells

### Domänen

Seien  $D_1, D_2, \dots, D_n$  Domänen (=Wertebereiche).

### Relationen

$R \subseteq D_1 \times \dots \times D_n$ .

- **Beispiel:**  $\text{telephoneBook} \subseteq \text{string} \times \text{string} \times \text{integer}$
- Wertebereiche dürfen identisch sein:  $D_i = D_j$  für  $i \neq j$ .
- Basierend auf Mengen.

### Relationenschema

- Legt die Struktur der gespeicherten Daten fest.
- Wird mit  $\text{sch}(R)$  oder  $R$  bezeichnet.
- **Notation:**  $R(A_1 : D_1, A_2 : D_2, \dots)$  mit  $A_i$  für Attribute.
- **Beispiel:**  $\text{telephoneBook}(\text{name} : \text{string}, \text{street} : \text{string}, \text{phoneNumber} : \text{integer})$

### Beispiele für zulässige Relationen

Gegeben

- Domänen  $D_1 = \{1, 2, 3\}$ ,  $D_2 = \{x, y, z\}$ ,  $D_3 = \{\alpha, \beta, \gamma, \omega\}$
- Attribute auf diesen Domänen  $A : D_1$ ,  $B : D_2$ ,  $C : D_3$

Welche der folgenden Relationen sind zulässig?

A	B	C
1	y	$\alpha$
3	x	$\alpha$
3	y	$\alpha$

zulässig

A	B	C
$\alpha$	y	$\alpha$
3	x	$\alpha$
2	y	$\beta$

nicht zulässig

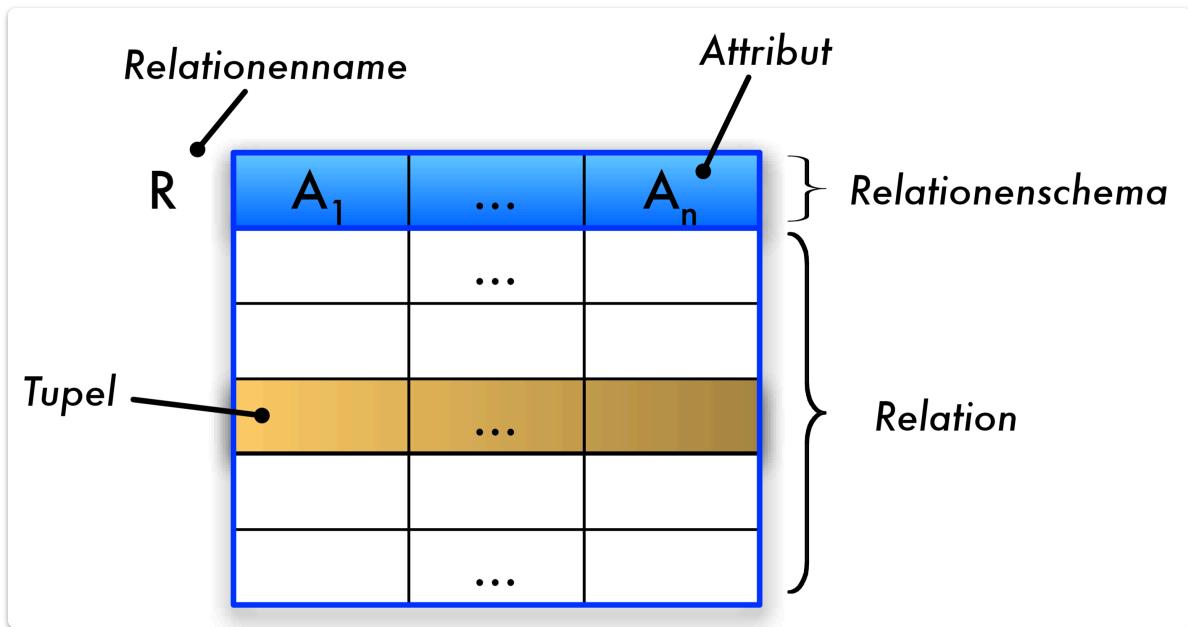
A	B	C
1	y	$\alpha$
3	x	$\alpha$
1	y	$\alpha$

nicht zulässig

## Veranschaulichung der Grundbegriffe

- **Fettgeschriebene Kopfzeile:** Relationenschema
- **Spaltenüberschrift:** Attribut
- **Weitere Einträge in der Tabelle:** Relation

- Eine Zeile der Tabelle: Tupel
- Ein Eintrag: Attributwert
- Unterstrichenes Attribut: Primärschlüssel



*Theorie vs. Realität:* In der Realität meist keine Unterscheidung zwischen Ausprägung (Instanz) und Schema einer Relation.

## Relationenmodell

wines	wineID	name	color	year	vineyard → producer
1042	La Rose Grand Cru	red	1998	Château La Rose	
2168	Creek Shiraz	red	2003	Creek	
3456	Zinfandel	red	2004	Helena	
2171	Pinot Noir	red	2001	Creek	
3478	Pinot Noir	red	1999	Helena	

producer	vineyard	area	region
Creek	Barossa Valley	South Australia	
Helena	Napa Valley	California	
Château La Rose	Saint-Emilion	Bordeaux	
Château La Pointe	Pomerol	Bordeaux	

## Fremdschlüssel

- Eine Relation kann die Primärschlüsselattribute einer anderen Tabelle beinhalten.
- Werte der Fremdschlüsselattribute müssen im Primärschlüssel der referenzierten Tabelle vorkommen. (Dies stellt die referentielle Integrität sicher, d.h., dass Verweise auf nicht existierende Einträge vermieden werden.)

## Eigenschaften des Relationalen Modells

### Tupelreihenfolge

Tupel einer Relation haben keine **Reihenfolge**.

name	age
Pat	1
Fred	2
Sue	3
Pam	4

name	age
Sue	3
Pam	4
Fred	2
Pat	1

Diese Relationen beinhalten dieselben Informationen. (Die Reihenfolge, in der Zeilen in einer Datenbanktabelle gespeichert oder abgerufen werden, ist irrelevant.)

## Attributreihenfolge

Die **mathematische Definition** von Tupeln sieht eine **bestimmte Reihenfolge** der Attribute des Tupels/der Relation vor.

name	age
Pat	1
Fred	2
Sue	3
Pam	4

age	name
1	Pat
2	Fred
3	Sue
4	Pam

Aber...

- Die Reihenfolge der Attribute ist in den meisten Anwendungen bedeutungslos.
- Das Verwenden von Attributnamen statt einer bestimmten Reihenfolge ist praktischer. (Man spricht Attribute über ihren Namen an, nicht über ihre Position.)
- Das kartesische Produkt wird kommutativ. ( $A \times B$  ist in der Praxis dasselbe wie  $B \times A$  bei Verwendung von Attributnamen.)

## Atomare Werte

- Werte eines Tupels sind **atomar** (unteilbar).
- Ein Wert kann kein zusammengesetzter Datentyp (Liste, Array, ...) oder eine Relation sein.

name	age
Pat	1
Fred	2
Sue	3
Pam	4

Alle Werte sind atomar

name			age
	Pat	Jensen	1
Fred	D.	Roosevelt	2
Sue	H.M.I.	Knuth	3
Pam	C.	Anderson	4

name ist nicht atomar

*Alle Werte sind atomar* vs. *name ist nicht atomar* (da "H.M. Knuth" oder "C. Anderson" aus mehreren Teilen bestehen könnten, was in der mathematischen Definition nicht atomar wäre.)

## Null Werte

Ein spezieller **Null** Wert wird verwendet, um unbekannte bzw. für gewisse Tupel unanwendbare Werte zu repräsentieren.

name	age
Pat	1
Fred	2
Sue	null
Pam	null

name	age
Pat	1
Fred	2
Sue	⊥
Pam	⊥

Alternative Notation

## Duplikate

Eine Relation folgt der **mathematischen Definition einer Menge**.

name	age
Pat	1
Fred	2
Sue	3
Pam	4

zulässig

name	age
Pat	1
Fred	2
Sue	3
Sue	3

unzulässig

Keine zwei Tupel einer Relation dürfen identische Werte in allen Attributen beinhalten. (Jedes Tupel muss einzigartig sein.)

## Beispiele

Welche der folgenden Relationen sind zulässig?

name	age
Pat	1
Fred	2
Sue	⊥
Sue	3

gültig

name	age
Pat	1
Fred	2
Sue	⊥
Sue	⊥

gültig

# Relationale Algebra

## Die Operatoren der relationalen Algebra

---

- Projektion  $\pi$
- Selektion  $\sigma$
- Umbenennung  $\rho$
- Kreuzprodukt  $\times$
- Vereinigung  $\cup$
- Differenz  $-$
- Schnitt  $\cap$
- Join (Verbund)  $\bowtie$
- Linker äußerer Join  $\bowtie_l$
- Rechter äußerer Join  $\bowtie_r$
- Äußerer Join \fullouterjoin
- Semi-Join (linker)  $\ltimes$
- Semi-Join (rechter)  $\rtimes$
- Gruppierung  $\gamma$
- Division  $\div$

## Basisoperatoren

---

- Projektion  $\pi$
- Selektion  $\sigma$
- Umbenennung  $\rho$
- Kreuzprodukt  $\times$
- Vereinigung  $\cup$
- Differenz  $-$

## Basisoperatoren

Jede Anfrage in relationaler Algebra kann ausschließlich mit Basisoperatoren ausgedrückt werden.

Das Weglassen eines Basisoperators verringert die Ausdruckskraft. (Man kann dann nicht mehr alle möglichen Anfragen formulieren.)

## Unäre vs. binäre Operatoren

- **Unäre Operatoren:**  $\sigma, \pi, \rho$  (Arbeiten auf einer einzelnen Relation.)
- **Binäre Operatoren:**  $\times, \cup, -$  (Arbeiten auf zwei Relationen.)

## Auswertung der Operatoren

---

## Operatoren und deren Verwendung

- Input: eine oder mehrere Relationen
- Output: eine Relation
- Operatoren können (nach bestimmten Regeln) kombiniert werden.

## Projektion

$\pi_{name, dep\_name}(instructor)$

instructor			
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000

Das Resultat ist eine Relation mit  $n$  Spalten, die durch das Weglassen der nicht angegebenen Spalten entsteht.

$\pi_{name, dep\_name}(instructor)$	
name	dep_name
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music
Einstein	Physics

## Erweiterte Projektion

$\pi_{ID, name, dep\_name, salary \div 12}(instructor)$

Vorher:

instructor			
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

Nacher:

instructor			
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	5417
12121	Wu	Finance	7500
15151	Mozart	Music	3333
22222	Einstein	Physics	7917
32343	El Said	History	5000
33456	Gold	Physics	7250

## Selektion

- Symbol:  $\sigma_F$
- Selektionsprädikat  $F$  besteht aus:

*Logischen Operatoren:  $\vee$  (oder),  $\wedge$  (und),  $\neg$  (nicht)*

Arithmetischen Vergleichsoperatoren:  $<, \leq, =, \neq, \geq, >$

\* ...und natürlich aus Attributnamen der Argumentrelation oder Konstanten als Operanden.

Auswahl von Zeilen einer Tabelle anhand eines Selektionsprädikats.

$$\sigma_{\text{salary} > 80000}(\text{instructor})$$

instructor			
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

$\sigma_{\text{salary} > 80000}(\text{instructor})$			
ID	name	dept_name	salary
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

## Umbenennung ( $\rho$ )

Die Umbenennungsoperation dient dazu, Namen von Relationen oder Attributen zu ändern. Dies ist besonders nützlich, um Eindeutigkeit in komplexen Abfragen zu gewährleisten oder die Lesbarkeit zu verbessern. In der Literatur wird auch oft das Symbol  $\beta$  verwendet.

### Umbenennung einer Relation

- Syntax:  $\rho_S(R)$

- **Erklärung:** Die Relation  $R$  erhält den neuen Namen  $S$ .
  - **Beispiel:** Eine Relation namens `Kunden` könnte in `Besteller` umbenannt werden, um in einem bestimmten Kontext verständlicher zu sein.

## Umbenennung von Spalten (Attributen)

- **Syntax:**  $\rho_{A \leftarrow B}(R)$
- **Erklärung:** Das Attribut  $B$  in der Relation  $R$  wird in  $A$  umbenannt.
  - **Beispiel:** In einer Relation `Produkte` könnte das Attribut `ProdNr` in `Produktnummer` umbenannt werden.

## Kartesisches Produkt (Kreuzprodukt) ( $\times$ )

Das Kartesische Produkt kombiniert alle möglichen Tupel (Zeilen) aus zwei Relationen miteinander. Es erzeugt eine neue Relation, die alle Tupelpaare aus den ursprünglichen Relationen enthält.

- **Syntax:**  $(R \times S)$
- **Erklärung:** Das Kreuzprodukt zweier Relationen  $R$  und  $S$  besteht aus allen möglichen Kombinationen von Tupeln. Die Anzahl der resultierenden Tupel ist das Produkt der Anzahlen der Tupel in  $R$  und  $S$ , also  $|R| \times |S|$  Paare.
- **Schema des Resultats:**
  - $sch(R \times S) = sch(R) \cup sch(S) = R \cup S$
  - Das Schema der Ergebnisrelation ist die Vereinigung der Schemata der beiden Ausgangsrelationen.
- **Wichtiger Hinweis:** Das Kartesische Produkt enthält oft viele (auch viele unsinnige) Kombinationen. Es ist eine grundlegende Operation, die häufig als Basis für Joins verwendet wird, bei denen dann zusätzliche Bedingungen angewendet werden, um nur die relevanten Kombinationen zu erhalten.
- **Referenzierung von Attributen:** Beim Referenzieren der Attribute des resultierenden Schemas wird  $R.A$  und  $S.A$  verwendet. Dies ist besonders wichtig bei Überlappungen der einzelnen Schemata (d.h. wenn beide Relationen Attribute mit dem gleichen Namen haben), um Unklarheiten zu vermeiden, welches Attribut gemeint ist.
  - **Beispiel:** Wenn sowohl Relation `Kunden` als auch Relation `Bestellungen` ein Attribut `ID` haben, würde man im Kreuzprodukt `Kunden.ID` und `Bestellungen.ID` verwenden, um sie zu unterscheiden.

## Beispiel

$$\pi_{V1.course}(\sigma_{V2.successor=5216 \wedge V1.successor=V2.course}$$

$$(\rho_{V1}(prerequisite) \times \rho_{V2}(prerequisite)))$$

prerequisite	
course	successor
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

V1.course	V1.successor	V2.course	V2.successor
5001	5041	5041	5216

V1.course
5001

Vorgänger zweiter Stufe der Vorlesung mit  
Nummer 5216

## Relationale Algebra: Mengenoperationen

Die grundlegenden Mengenoperationen **Vereinigung**, **Schnitt** und **Differenz** können auch auf Relationen angewendet werden.

### Voraussetzung für Mengenoperationen: Vereinigungskompatibilität

Damit Mengenoperationen auf zwei Relationen  $R$  und  $S$  angewendet werden können, müssen diese **vereinigungskompatibel** sein. Das bedeutet:

- **Gleiche Anzahl an Attributen:** Beide Relationen müssen die gleiche Anzahl von Spalten (Attributen) besitzen.
- **Gleicher Wertebereich in Spaltenreihenfolge:** Für jedes Attribut in der jeweiligen Spaltenreihenfolge müssen die Wertebereiche (Datentypen) gleich sein. Das heißt, wenn das erste Attribut von  $R$  ein `Integer` ist, muss das erste Attribut von  $S$  ebenfalls ein `Integer` sein und so weiter für alle Attribute.

### Vereinigung ( $\cup$ )

Die Vereinigung von zwei Relationen  $R$  und  $S$  sammelt alle Tupel (Zeilen) aus beiden Relationen und entfernt dabei Duplikate. Das Ergebnis ist eine neue Relation, die alle einzigartigen Tupel enthält, die entweder in  $R$  oder in  $S$  (oder in beiden) vorkommen.

- **Syntax:**  $(R \cup S)$
- **Beispiel:** Eine Vereinigung aller Abteilungsnamen von Instruktoren und Studenten.

$$\pi_{name,dep\_name}(instructor) \cup \pi_{name,dep\_name}(student)$$

### Differenz (Ohne, Minus) ( $-$ oder $\setminus$ )

Die Differenz von zwei Relationen  $R$  und  $S$  eliminiert alle Tupel aus der ersten Relation ( $R$ ), die auch in der zweiten Relation ( $S$ ) vorkommen. Das Ergebnis ist eine neue Relation, die nur die Tupel enthält, die exklusiv in  $R$  sind und nicht in  $S$  vorkommen.

- **Syntax:**  $(R - S)$  bzw.  $(R \setminus S)$
- **Beispiel:** Angenommen, wir haben zwei Relationen `instructor_departments` und `student_departments`, die jeweils die Abteilungsnamen enthalten, in denen Instruktoren bzw. Studenten tätig sind.

instructor_departments	student_departments
dept_name	dept_name
Comp. Sci.	Comp. Sci.
Music	History
History	Finance
Biology	Physics
Elec. Eng.	Music

`instructor_departments – student_departments`

- Die Differenz würde die Abteilungsnamen zurückgeben, in denen Instruktoren tätig sind, aber keine Studenten.

dept_name
Biology
Elec. Eng.

## Überblick über die Basisoperatoren

Ausdruck	Schema	Arität	Min. Kardinalität	Max. Kardinalität
$\sigma_F(R)$	$\mathcal{R}$	$ \mathcal{R} $	0	$ \mathcal{R} $
$\pi_L(R)$	$L$	$\leq  \mathcal{R} $	0	$ \mathcal{R} $
$R \cup S$	$\mathcal{R} (= \mathcal{S})$	$ \mathcal{R}  (=  \mathcal{S} )$	$\max( \mathcal{R} ,  \mathcal{S} )$	$ \mathcal{R}  +  \mathcal{S} $
$R - S$	$\mathcal{R} (= \mathcal{S})$	$ \mathcal{R}  (=  \mathcal{S} )$	0	$ \mathcal{R} $
$R \times S$	$\mathcal{R} \circ \mathcal{S}$	$ \mathcal{R}  +  \mathcal{S} $	$ \mathcal{R}  \cdot  \mathcal{S} $	$ \mathcal{R}  \cdot  \mathcal{S} $
$\rho_S(R)$	$\mathcal{R}$	$ \mathcal{R} $	$ \mathcal{R} $	$ \mathcal{R} $

## Schnitt (Durchschnitt) ( $\cap$ )

Der Schnitt von zwei Relationen  $R$  und  $S$  besteht aus der Menge aller Tupel, die in beiden Relationen gemeinsam vorkommen. Im Gegensatz zur Vereinigung und Differenz ist der **Schnitt kein Basisoperator** in der relationalen Algebra, da er sich aus den Basisoperatoren (Differenz) ableiten lässt.

- **Syntax:**  $(R \cap S)$

- **Erklärung:** Der Schnitt liefert alle Tupel, die sowohl in Relation  $R$  als auch in Relation  $S$  enthalten sind.
- **Ableitung aus Differenz:** Der Schnitt kann wie folgt als Differenz ausgedrückt werden:
  - $(R \cap S = R - (R - S))$
  - **Intuitive Erklärung:** Nimm alle Tupel aus  $R$ . Ziehe davon alle Tupel ab, die *nicht* in  $S$  sind (also die Tupel, die nur in  $R$  vorkommen). Was übrig bleibt, sind genau die Tupel, die sowohl in  $R$  als auch in  $S$  sind.
- **Beispiel:** Angenommen, wir haben weiterhin die Relationen `instructor_departments` und `student_departments`.

<code>instructor_departments</code>	<code>student_departments</code>
dept_name	dept_name
Comp. Sci.	Comp. Sci.
Music	History
History	Finance
Biology	Physics
Elec. Eng.	Music

`instructor_departments`  $\cap$  `student_departments`

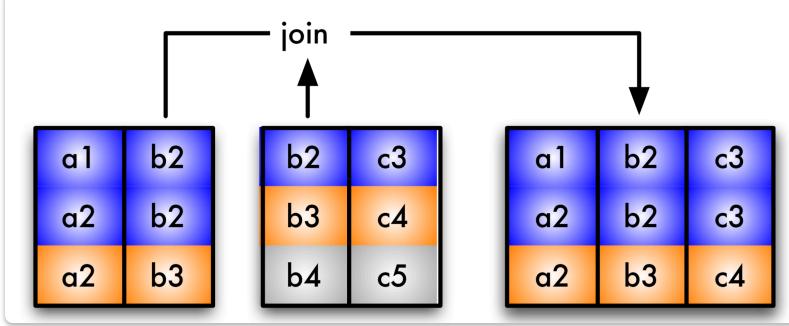
- Der Schnitt würde die Abteilungsnamen zurückgeben, in denen sowohl Instruktoren als auch Studenten tätig sind.

dept_name
Comp. Sci.
Music
History

## Natürlicher Verbund (Natural Join) ( $\bowtie$ )

Der natürliche Verbund ist eine wichtige Operation, um Informationen aus zwei oder mehr Relationen basierend auf gemeinsamen Attributnamen zu kombinieren. Er verknüpft Tabellen (Relationen) über **gleichbenannte Spalten** und verschmilzt jeweils zwei Tupel, wenn sie dort **gleiche Werte** aufweisen.

- **Gegeben zwei Relationen (+ Schemata):**
  - $R(A_1, \dots, A_m, B_1, \dots, B_k)$
  - $S(B_1, \dots, B_k, C_1, \dots, C_n)$



- **Erklärung zum Bild:** Die Relationen  $R$  und  $S$  haben gemeinsame Attribute  $B_1, \dots, B_k$ . Der Natural Join sucht Tupelpaare, bei denen die Werte dieser gemeinsamen Attribute übereinstimmen, und kombiniert diese Tupel zu einem neuen, breiteren Tupel. Die gemeinsamen Attribute erscheinen im Ergebnis nur einmal.
- **Der natürliche Join kann durch ein Kreuzprodukt gefolgt von Selektionen und Projektionen ausgedrückt werden.** Dies zeigt, dass der Natural Join kein fundamentaler Basisoperator ist, sondern eine abgeleitete Operation, die bequem ist, aber mit grundlegenderen Operationen simuliert werden kann.
  - **Syntax:**  $R \bowtie S = \pi_{A_1, \dots, A_m, R.B_1, \dots, R.B_k, C_1, \dots, C_n} (\sigma_{R.B_1=S.B_1 \wedge \dots \wedge R.B_k=S.B_k} (R \times S))$
  - **Intuitive Erklärung der Ableitung:**
    1. **Kartesisches Produkt ( $R \times S$ ):** Zuerst werden alle möglichen Kombinationen von Tupeln aus  $R$  und  $S$  gebildet. Das Ergebnis enthält dann doppelte Spalten für die gemeinsamen Attribute (z.B.  $R.B_1$  und  $S.B_1$ ).
    2. **Selektion ( $\sigma_{...}$ ):** Anschließend werden nur die Tupel ausgewählt, bei denen die Werte der gleichbenannten Attribute aus  $R$  und  $S$  übereinstimmen (z.B.  $R.B_1 = S.B_1$ ,  $R.B_2 = S.B_2$ , usw.). Dies filtert die "sinnvollen" Kombinationen heraus.
    3. **Projektion ( $\pi_{...}$ ):** Zuletzt werden die gewünschten Attribute projiziert. Dabei werden die doppelten Spalten der gemeinsamen Attribute entfernt, sodass jedes gemeinsame Attribut nur einmal im Endergebnis erscheint.
- **Schema des Resultats:**

$$R \bowtie S = \pi_{A_1, \dots, A_m, R.B_1, \dots, R.B_k, C_1, \dots, C_n} (\sigma_{R.B_1=S.B_1 \wedge \dots \wedge R.B_k=S.B_k} (R \times S))$$

$R \bowtie S$											
$R - S$				$R \cap S$				$S - R$			
$A_1$	$A_2$	$\dots$	$A_m$	$B_1$	$B_2$	$\dots$	$B_k$	$C_1$	$C_2$	$\dots$	$C_n$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

- Das resultierende Schema enthält alle Attribute aus  $R$  und  $S$ , wobei die gemeinsamen Attribute nur einmal auftauchen.
- **Beispiel Schema:**  $R \bowtie S$  hätte die Attribute  $(A_1, \dots, A_m, B_1, \dots, B_k, C_1, \dots, C_n)$ .
- **Reihenfolge der Attribute:** Die Reihenfolge der Attribute in der Ergebnisrelation wird durch die Attribute der gegebenen Relationen bestimmt. Das heißt, das resultierende Schema ist nicht unbedingt wie im obigen Beispiel sortiert (z.B. zuerst alle Attribute von  $R$ , dann alle von  $S$ ), sondern kann auch anders angeordnet sein, je nach Implementierung.

## Beispiel

wines  $\bowtie$  producer

wines	wineID	name	color	year	vineyard
	1042	La Rose Grand Cru	red	1998	Château La Rose
	2168	Creek Shiraz	red	2003	Creek
	2171	Pinot Noir	red	2001	Creek
	4711	Riesling Reserve	white	1999	Müller

producer		vineyard	area	region	
	Creek	Barossa Valley	South Australia		
	Helena	Napa Valley	California		
	Château La Rose	Saint-Emilion	Bordeaux		

Resultat:

wineID	name	...	vineyard	...	region
1042	La Rose Grand Cru	...	Ch. La Rose	...	Bordeaux
2168	Creek Shiraz	...	Creek	...	South Australia
2171	Pinot Noir	...	Creek	...	South Australia

Tupel, die keinen Partner finden (*dangling tuples*), werden “eliminiert”.

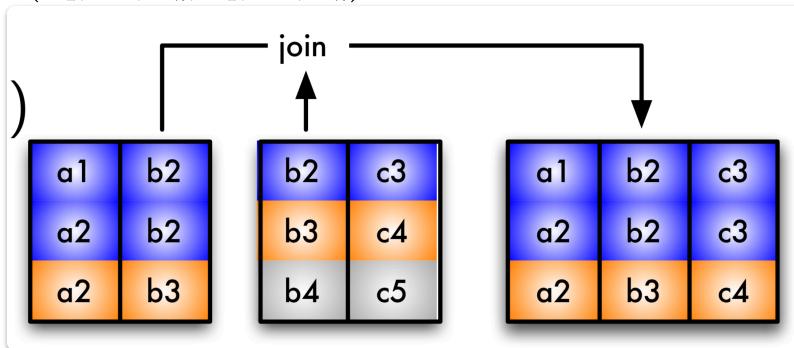
## Join Kommutativität

Eine wichtige Eigenschaft von Operationen ist die Kommutativität. Bei Joins stellt sich die Frage, ob die Reihenfolge der Operanden das Ergebnis beeinflusst.

Ist  $R \bowtie S = S \bowtie R$  wahr oder falsch?

- Gegeben zwei Relationen (+ Schema):

- $R(A_1, \dots, A_m, B_1, \dots, B_k)$
- $S(B_1, \dots, B_k, C_1, \dots, C_n)$



- Die Reihenfolge beeinflusst nicht "wirklich" das Resultat:

- Die Reihenfolge der Attribute im *Schema* der Ergebnisrelation ist unterschiedlich.
- Aber der "Inhalt" der Tupel und Relationen ist "gleich". Das bedeutet, die Menge der Informationen, die durch die Verknüpfung gewonnen wird, ist identisch, unabhängig davon, ob man  $R$  mit  $S$  oder  $S$  mit  $R$  verknüpft. Die Tupel sind dieselben, nur die Reihenfolge der Spalten kann variieren.

- Betrachtung der Kommutativität in der Praxis:

- Momentan (im Kontext der Basiskonzepte) werden Joins und Kreuzprodukte *nicht als kommutativ* betrachtet, wenn man streng die Reihenfolge der Attribute im

Ergebnis beachtet.

- Bei der Anfrageoptimierung (ein späteres Thema in der Lehrveranstaltung) werden natürliche Joins sowie Kreuzprodukte und andere Join-Varianten jedoch *als kommutativ* behandelt. Dies ist entscheidend, um die Reihenfolge der Join-Operationen zu optimieren und so die Effizienz von Datenbankabfragen zu verbessern.
  - Exakte mathematische Definition:
    - Wenn wir die mathematische Definition von Tupeln (die eine feste Reihenfolge von Elementen haben) und Joins (die eine feste Reihenfolge der Attribute im Schema haben) einhalten und trotzdem Kommutativität annehmen wollen, müssen wir eine **Projektion vornehmen, um die Attribute entsprechend zu sortieren**.
    - $\pi_L(R \bowtie S) = \pi_L(S \bowtie R)$
    - Wobei  $L$  eine Liste der Attribute ist, die eine einheitliche Sortierung für beide Ergebnisse sicherstellt. Dies bedeutet, dass die Tupelmengen nach der Projektion identisch sind, auch wenn die ursprünglichen Schemata der Joins unterschiedliche Attributreihenfolgen hatten.
-

# Relationale Algebra: Zusätzliche Join-Varianten

Neben dem Natural Join gibt es weitere Verbundoperationen, die spezifische Anforderungen an die Verknüpfung von Relationen erfüllen.

## Theta-Join ( $\theta$ -Join oder allgemeiner Join)

Der Theta-Join ist eine sehr flexible Join-Operation, die zwei Relationen basierend auf einem beliebigen Prädikat (Bedingung) verknüpft, das die beteiligten Attribute betrifft.

- Gegeben zwei Relationen (+ Schemata):

- $R(A_1, \dots, A_n)$
- $S(B_1, \dots, B_m)$
- $\theta$  ist ein beliebiges Prädikat über den beteiligten Attributen (z.B.  $R.A_i > S.B_j$ ,  $R.A_i \neq S.B_j$ , etc.).

- Syntax:  $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

- Erklärung: Der Theta-Join wird ausgedrückt als ein **Kartesisches Produkt** ( $R \times S$ ) gefolgt von einer **Selektion** ( $\sigma_{\theta}$ ), die alle Tupel aus dem Kreuzprodukt herausfiltert, für die das angegebene Prädikat  $\theta$  zutrifft.

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

$R \bowtie_{\theta} S$							
$\mathcal{R}$				$\mathcal{S}$			
$A_1$	$A_2$	$\dots$	$A_n$	$B_1$	$B_2$	$\dots$	$B_m$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

- Das Schema des Theta-Joins ist einfach die Konkatenation der Schemata von  $R$  und  $S$ , da keine Attribute aufgrund von Gleichheit verschmolzen werden wie beim Natural Join.

## Equi-Join

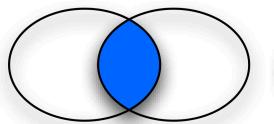
Der Equi-Join ist ein Spezialfall des Theta-Joins, bei dem das Join-Prädikat  $\theta$  **nur Gleichheit** ( $=$ ) prüft.

- Erklärung: Anstatt eines beliebigen Vergleichsoperators ( $<$ ,  $>$ ,  $\neq$ , etc.) verwendet der Equi-Join ausschließlich den Gleichheitsoperator ( $=$ ) für die Verknüpfungsbedingung zwischen den Attributen der Relationen. Der Natural Join ist ein Spezialfall des Equi-Joins, der zusätzlich die doppelten Attribute nach dem Join entfernt.

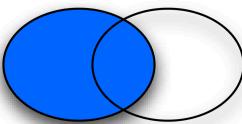
## Äußere Verbunde (Outer Joins)

Äußere Verbunde sind wichtig, wenn man Tupel erhalten möchte, die **keinen Joinpartner** in der anderen Relation finden. Diese Tupel werden als "partnerlose" Tupel bezeichnet. Normalerweise würden diese Tupel bei einem (inneren) Join verloren gehen. Outer Joins füllen die fehlenden Attribute dieser partnerlosen Tupel mit Nullwerten auf.

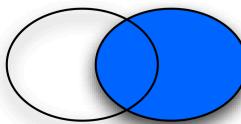
- ☒ Left outer Join (Linker äußerer Verbund):  
auch "partnerlose" Tupel der linken Relation bleiben erhalten
- ☒ Right outer Join (Rechter äußerer Verbund):  
auch "partnerlose" Tupel der rechten Relation bleiben erhalten
- ☒ (Full) outer Join (Vollständiger äußerer Verbund):  
die "partnerlosen" Tupel beider Relationen bleiben erhalten



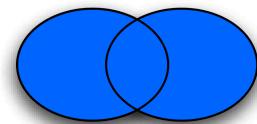
natural join



left outer join



right outer join



full outer join

### Natural Join (Natürlicher Verbund)

<b>L</b>			<b>R</b>			<b>Result</b>				
A	B	C	C	D	E	A	B	C	D	E
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$					

### Left Outer Join (Linker äußerer Verbund)

<b>L</b>			<b>R</b>			<b>Result</b>				
A	B	C	C	D	E	A	B	C	D	E
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$	$a_2$	$b_2$	$c_2$	$\perp$	$\perp$

## Right Outer Join (Rechter äußerer Verbund)

L			R			Result				
A	B	C	C	D	E	A	B	C	D	E
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$	$\perp$	$\perp$	$c_3$	$d_2$	$e_2$

## Outer Join (Äußerer Verbund)

L			R			Result				
A	B	C	C	D	E	A	B	C	D	E
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$a_1$	$b_1$	$c_1$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$	$a_2$	$b_2$	$c_2$	$\perp$	$\perp$
			$\perp$	$\perp$	$c_3$	$\perp$	$\perp$	$d_2$	$e_2$	

## Semi-Joins

Semi-Joins sind spezielle Join-Varianten, die dazu dienen, Tupel aus einer Relation zu finden, die einen Joinpartner in einer anderen Relation haben, aber ohne die Attribute der Joinpartner-Relation dem Ergebnis hinzuzufügen. Sie sind nützlich, wenn man nur wissen möchte, welche Tupel aus der Ausgangsrelation "passen", ohne die kombinierten Informationen.

### Linker Semi-Join ( $\ltimes$ )

- **Definition:** Finde alle Tupel der **linken Relation**, die Joinpartner in der **rechten Relation** haben.
- **Syntax:**  $L \ltimes R = \pi_L(L \bowtie R)$ 
  - **Erklärung:** Ein linker Semi-Join zwischen Relation  $L$  und  $R$  kann ausgedrückt werden als eine Projektion ( $\pi_L$ ) des Ergebnisses eines Natural Joins ( $L \bowtie R$ ) zurück auf die Attribute der linken Relation  $L$ .
  - $L$  repräsentiert hier die Menge der Attribute der linken Relation, die im Ergebnis erscheinen sollen. Da es ein Semi-Join ist, sind dies alle Attribute der linken Relation.
- **Beispiel Linker Semi-Join ( $L \ltimes R$ ):**

L			R			Result		
A	B	C	C	D	E	A	B	C
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$a_1$	$b_1$	$c_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$			

- **Ergebnis ( $L \ltimes R$ ):** Es werden nur die Tupel aus L ausgegeben, für die es eine Übereinstimmung in R gibt (hier über das Attribut C). Die Spalten von R werden nicht im Ergebnis aufgeführt.

## Rechter Semi-Join ( $\bowtie$ )

- **Definition:** Finde alle Tupel der **rechten Relation**, die Joinpartner in der **linken Relation** haben.
- **Syntax:**  $L \bowtie R = \pi_R(L \bowtie R)$ 
  - **Erklärung:** Ein rechter Semi-Join zwischen Relation  $L$  und  $R$  kann ausgedrückt werden als eine Projektion ( $\pi_R$ ) des Ergebnisses eines Natural Joins ( $L \bowtie R$ ) zurück auf die Attribute der rechten Relation  $R$ .
  - $R$  repräsentiert hier die Menge der Attribute der rechten Relation, die im Ergebnis erscheinen sollen.
- **Beispiel Rechter Semi-Join ( $L \bowtie R$ ):**

<b>L</b>			<b>R</b>			<b>Result</b>		
A	B	C	C	D	E	C	D	E
$a_1$	$b_1$	$c_1$	$c_1$	$d_1$	$e_1$	$c_1$	$d_1$	$e_1$
$a_2$	$b_2$	$c_2$	$c_3$	$d_2$	$e_2$			

$\times$       =

- **Ergebnis ( $L \bowtie R$ ):** Es werden nur die Tupel aus  $R$  ausgegeben, für die es eine Übereinstimmung in  $L$  gibt (hier über das Attribut C). Die Spalten von  $L$  werden nicht im Ergebnis aufgeführt.

# Gruppierung und Aggregation

Die Gruppierung und Aggregation sind mächtige Operationen, um Daten in einer Relation zusammenzufassen und Berechnungen auf diesen zusammengefassten Daten durchzuführen.

## Gruppierung

---

Tupel mit gleichen Attributwerten (für eine angegebene Liste von Attributen) werden **gruppiert**. Das bedeutet, die Relation wird in Gruppen unterteilt, wobei alle Tupel innerhalb einer Gruppe dieselben Werte für die angegebenen Gruppierungsattribute haben.

## Aggregation

---

Auf jede dieser gebildeten Gruppen wird anschließend eine **Aggregatfunktion** angewendet. Diese Funktionen berechnen einen einzelnen Wert für jede Gruppe, der die zusammengefassten Daten dieser Gruppe repräsentiert.

### Typische Aggregatfunktionen:

- **count:** Zählt die Anzahl der Elemente (Tupel) pro Gruppe.
- **sum:** Berechnet die Summe der Werte eines bestimmten Attributs pro Gruppe.
- **min, max, avg:** Berechnen das Minimum, Maximum bzw. den Durchschnittswert eines bestimmten Attributs pro Gruppe.

### Notation:

- $\mathcal{G}_{L,F}(R)$ 
  - $L$ : Dies ist eine Liste der Attribute, nach denen die Gruppierung erfolgen soll (die "GROUP BY"-Attribute in SQL).
  - $F$ : Dies ist die Aggregatfunktion, die auf jede Gruppe angewendet wird (z.B. `count(*)`, `sum(Gehalt)`, `avg(Alter)`).
  - Alternative Symbole  $\mathcal{G}$  oder  $\beta$  können ebenfalls für diese Operation verwendet werden.

### Beispiele

## Die Anzahl der Studierenden pro Semester

$$\gamma_{\text{semester}; \text{count}(*)}(\text{student})$$

student		
studID	name	semester
24002	Nielsen	18
25403	Hansen	12
26120	Pedersen	10
26830	Andersen	6
27550	Larsen	6

$\gamma_{\text{semester}; \text{count}(*)}(\text{student})$	
semester	count(*)
18	1
12	1
10	1
6	2

Die Anzahl der Studierenden pro Semester sowie  
die minimale studID pro Gruppe

$$\gamma_{\text{semester}; \text{count}(*), \text{min}(\text{studID})}(\text{student})$$

student		
studID	name	semester
24002	Nielsen	18
25403	Hansen	12
26120	Pedersen	10
26830	Andersen	6
27550	Larsen	6

$\gamma_{\text{semester}; \text{count}(*), \text{min}(\text{studID})}(\text{student})$		
semester	count(*)	min(studID)
18	1	24002
12	1	25403
10	1	26120
6	2	26830

Aggregatfunktionen werden auch auf Duplikaten evaluiert, um z.B. die  
Summe korrekt berechnen zu können!

count vs. count-distinct, sum vs. sum-distinct, etc.

# Relationale Division ( $\div$ )

Die relationale Division ist eine Operation, die alle Tupel aus einer Relation findet, die mit *allen* Tupeln einer anderen Relation in Beziehung stehen. Sie ist besonders nützlich für "für alle"-Anfragen.

- **Grundidee:** Finde Elemente (aus einer Menge von Werten in Relation R), die mit *allen* Elementen einer bestimmten Menge (aus Relation S) assoziiert sind.
- **Beispiel:** Finde die `studIDs` der Studierenden, die *alle* Vorlesungen mit 4 ECTS besuchen.
  - **Relation takes :** `(studID, courseID)` - welche Studierenden welche Vorlesungen belegen
  - **Relation course :** `(courseID, title, ects, teacher)` - Informationen zu Vorlesungen
  - **Gesuchte Vorlesungen (alle 4 ECTS-Vorlesungen):**  $\pi_{courseID}(\sigma_{ects=4}(course))$
- **Formale Definition:**

$$\bullet \quad R \div S = \pi_{R-S}(R) - \pi_{R-S}((\pi_{R-S}(R) \times S) - R)$$

- **Intuitive Erklärung der Definition:**

1. **Schritt 1: Projektion von R auf die Nicht-S-Attribute** ( $\pi_{R-S}(R)$ ): Dies sind die möglichen "Kandidaten" für das Ergebnis der Division (im Beispiel: die `studIDs`).
2. **Schritt 2: Kreuzprodukt der Kandidaten mit S** ( $\pi_{R-S}(R) \times S$ ): Für jeden Kandidaten wird eine "erwartete" Menge an Tupeln generiert, die alle Kombinationen des Kandidaten mit jedem Element aus S enthält. Dies repräsentiert, welche Kombinationen jeder Kandidat *haben sollte*, um alle Elemente von S zu "erfüllen".
3. **Schritt 3: Differenz (( $\pi_{R-S}(R) \times S$ ) - R):** Von diesen erwarteten Tupeln wird die ursprüngliche Relation R abgezogen. Das Ergebnis sind die Tupel, die ein Kandidat *nicht* hat, obwohl er sie haben *sollte*, um alle Elemente von S zu erfüllen.
4. **Schritt 4: Letzte Differenz ( $\pi_{R-S}(R) - \dots$ ):** Von den ursprünglichen Kandidaten (aus Schritt 1) werden diejenigen abgezogen, die in Schritt 3 als "nicht vollständig" identifiziert wurden. Was übrig bleibt, sind genau die Kandidaten, die *alle* erforderlichen Verbindungen zu S hatten.

- **Beispiel zur Veranschaulichung der Division:**

$$takes \div \pi_{courseID}(\sigma_{ects=4}(course))$$

takes	
studID	courselD
26120	5001
27550	5001
27550	4052
28106	4052
28106	4630
28106	5001
28106	5041
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

course			
courselD	title	ects	teacher
5001	DBS	4	2137
5041	Robotics	4	2125
5043	Software Engineering	3	2126
5049	Ethics	2	2125
4052	Logic	4	2125
5052	Theory of Science	3	2126
5216	Bioethics	2	2126
5259	Chemistry	2	2133
5022	Believe and Knowledge	2	2134
4630	Physics	4	2137

takes	
studID	courselD
26120	5001
27550	5001
27550	4052
28106	4052
28106	4630
28106	5001
28106	5041
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

$\pi_{courseID}(\sigma_{ects=4}(course))$
courselD
5001
5041
4052
4630

result
studID
28106

$R$	
$M$	$V$
$m_1$	$v_1$
$m_1$	$v_2$
$m_1$	$v_3$
$m_1$	$v_4$
$m_2$	$v_1$
$m_2$	$v_2$
$m_3$	$v_1$
$m_3$	$v_3$
$m_4$	$v_3$

$S_1$
$V$

$S_2$
$V$

$S_3$
$V$

$R \div S_1$
$M$

$R \div S_2$
$M$

$R \div S_3$
$M$

- Ein weiteres Beispiel (Wine Recommendation):

wineRecommendation

wine	reviewer
La Rose Grand Cru	Parker
Pinot Noir	Parker
Riesling Reserve	Parker
La Rose Grand Cru	Clarke
Pinot Noir	Clarke
Riesling Reserve	Gault-Millau

wineGuide1

reviewer
Parker
Clarke

wineGuide2

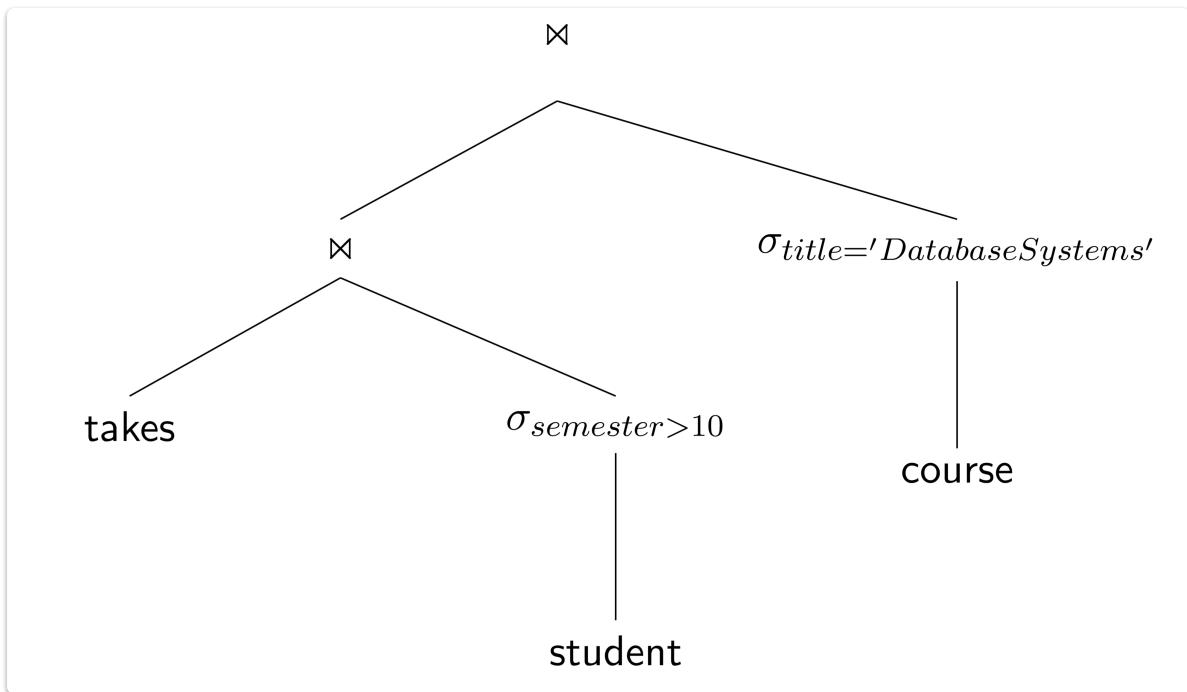
reviewer
Parker
Gault-Millau

$$wineRecommendation \div wineGuide2$$

wine
Riesling Reserve

# Operatorenbaumdarstellung

Die Operatorenbaumdarstellung ist eine visuelle Methode, um relationale Algebra-Ausdrücke darzustellen. Sie zeigt die Reihenfolge der Operationen und deren Abhängigkeiten. Die Blätter des Baumes sind die Relationen, und die inneren Knoten sind die relationalen Operatoren.



- **Erklärung:** In einem Operatorenbaum fließen die Daten von den Blättern (den ursprünglichen Relationen) nach oben zu den Wurzelknoten, wo die letzte Operation ausgeführt wird. Jeder innere Knoten repräsentiert eine Operation der relationalen Algebra (z.B. Projektion, Selektion, Join, Vereinigung etc.). Die Reihenfolge, in der die Operationen ausgeführt werden, ist hierarchisch von unten nach oben im Baum. Diese Darstellung ist besonders wichtig für die Anfrageoptimierung in Datenbanksystemen, da verschiedene Baumstrukturen (die das gleiche Ergebnis liefern) unterschiedliche Ausführungszeiten haben können.

## Einschränkungen der relationalen Algebra

- **Eingeschränkte Arithmetik**
  - Kann beispielsweise nicht direkt die Mietpreiserhöhung um 10% berechnen (z.B. "Finde den Mietpreis bei einer angenommenen Erhöhung von 10%").
- **Aggregatfunktionen fehlen in den Basisoperatoren der relationalen Algebra**
  - Fragen wie "Wie viele Filme hat jeder Kunde reserviert?" können nicht direkt beantwortet werden, da Funktionen wie SUM, COUNT, AVG fehlen.
- **Transitive Hülle kann nicht gebildet werden**
  - Beispiel: Für eine  $Part(Part, ConstituentPart)$  Relation ist es schwierig, alle Teile zu finden, aus denen ein Auto besteht, wenn es Unterteilungen gibt (z.B. Motor besteht

aus X, X aus Y, Y aus Z).

- **Kein Sortieren oder Ausgeben in verschiedenen Formaten**
  - Es ist nicht möglich, eine Zusammenfassung von Rechnungen nach Kundennamen sortiert anzuzeigen. Die relationale Algebra liefert lediglich unstrukturierte Mengen von Tupeln.
- **Keine Modifikationen an einer Relation möglich**
  - Operationen wie "Erhöhe alle 3.25 Preise auf 3.50" sind nicht vorgesehen. Die relationale Algebra ist primär für Abfragen gedacht, nicht für Datenmanipulation.

## Erweiterte Relationale Algebra

---

- Diese Einschränkungen werden in der **erweiterten relationalen Algebra** behoben.

## Zusammenfassung

- **Relationenmodell**
  - Keine Duplikate (mengenbasiert) - (Jedes Tupel in einer Relation ist einzigartig, wie in einer mathematischen Menge).
- **Relationale Algebra**
  - Jeder Operator erfordert Relationen als Input und erzeugt eine Relation als Output.
  - **Operatoren**
    - Sechs fundamentale Operatoren.
    - Viele abgeleitete Operatoren.
    - Operatoren erzeugen eine Abfragesprache.
    - Vereinigungskompatibilität (Relationen müssen die gleiche Anzahl und die gleichen Typen von Attributen haben, um vereint werden zu können).
  - **Verschiedene Join-Arten**
    - Es ist wichtig, die verschiedenen Join-Typen in der **Praxis** zu schätzen und zu verstehen, wenn man Abfragen an eine Datenbank stellt.
- Die relationale Algebra ist die Basis für **Anfragebearbeitung und -optimierung**.