

# Relationale Algebra

# Relationale Abfragesprachen/Relational Query Languages (QL)

- Abfragesprachen: Daten aus einer Datenbank zu manipulieren und abzufragen (retrieve information)
- Das relationale Modell hat einfache und leistungsfähige Abfragesprachen (man kann viel optimieren)
- Abfragesprache  $\neq$  Programmiersprache
- Abfragesprachen:
  - Nicht für komplexe Operationen
  - Erlaubt einfacher und effizienter Zugriff zu großen Datensätze

# Formale Relationale Abfragesprachen (Query Languages)

- Zwei mathematische Abfragesprachen stellen die theoretische Grundlage der „reellen“ Abfragesprachen (wie z.B. SQL) in relationalen Datenbanken:
  - Relationale Algebra:
    - kann Ausführungspläne beschreiben (operational)
  - Relationale Kalküle:
    - Der Benutzer kann beschreiben was er haben will und nicht wie es berechnet werden soll (non-operational, deklarativ)
    - Domänenkalkül, Tupelkalkül

# Relationale Algebra

- Fünf Basisoperationen:
  - **Projektion** (  $\pi$  ) : wählt bestimmte Spalten aus der Relation und gibt diese als neue Relation aus („löscht“ die anderen Spalten)
  - **Selektion** (  $\sigma$  ) : wählt bestimmte Zeilen aus der Relation und gibt diese als neue Relation aus („löscht“ die anderen Zeilen)
  - **Kartesisches Produkt** (  $\times$  ) : erlaubt die Verknüpfung zweier Relationen
  - **Differenz** (  $-$  ) : gibt die Tupeln aus der ersten Relation, die sich nicht in der zweiten Relation befinden, aus
  - **Vereinigung** (  $\cup$  ) : gibt die Tupeln aus der ersten und zweiten Relation aus
- Zusätzliche Operatoren: Umbenennen, Durchschnitt, Division, Verbund
- Die Operationen können zusammengesetzt sein (jede Operation hat eine Relation als Ergebnis)

# Projektion

- **Definition.** Sei  $L = (A_1, \dots, A_n)$  eine Teilmenge von Attributen(Spalten) aus der Relation  $R$ . Die Projektion der Attribute  $L$  einer Relation  $R$  ist definiert als die Relation  $R'(A_1, \dots, A_n)$  mit:

$$R' = \pi_L(R) = \{ t' \mid t \in R \wedge t'.A_1 = t.A_1 \wedge \dots \wedge t'.A_n = t.A_n \}$$

- Oder, anders gesagt:
  - die Projektion aus einem Tupel  $t \in R$  ist definiert als das Tupel

$$\pi_L(t) = (t(A_1), \dots, t(A_n))$$

- Die Projektion der Relation  $R$  ist definiert als die Relation

$$\pi_L(R) = \{ \pi_L(t) \mid t \in R \}$$

# Projektion - Beispiel

Studenten

<u>MatrikelNr</u>	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400



$\pi_{Name, Ort}(Studenten)$



Name	Ort
Schmidt	Würzburg
Meisel	Schweinfurt
Schmidt	Rimpar
Krause	Würzburg
Schäfer	Kitzingen
Rasch	Würzburg
Bakowski	Schweinfurt

# Projektion in SQL

- Ist  $\pi_{\text{Name,Ort}}(\text{Studenten})$  äquivalent mit

`SELECT Name, Ort FROM Studenten` ?

- **NEIN!**

- Relationale Algebra funktioniert mit Mengen  $\Rightarrow$  keine Duplikate(identische Tupeln)
- Das ist in SQL nicht standardmäßig so!
- Äquivalent:

`SELECT DISTINCT Name, Ort FROM Studenten`

# Selektion / Restriktion

- **Definition.** Die Selektion einer Relation R ist definiert als die Menge aller Tupel aus R, die der Selektionsbedingung P genügen:

$$\sigma_P(R) = \{ t \mid t \in R \wedge P(t) \}$$

- Die Bedingung P setzt sich zusammen aus:
  - Operanden: Konstanten oder Name eines Attributs
  - Vergleichsoperatoren: =,  $\neq$ , <,  $\leq$ , >,  $\geq$
  - Boolsche Operatoren:  $\vee$ ,  $\wedge$ ,  $\neg$



# Selektion - Beispiel

Studenten

<u>MatrikelNr</u>	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400



$\sigma_{Name='Schmidt'}(Studenten)$

<u>MatrikelNr</u>	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0

# Selektion in SQL

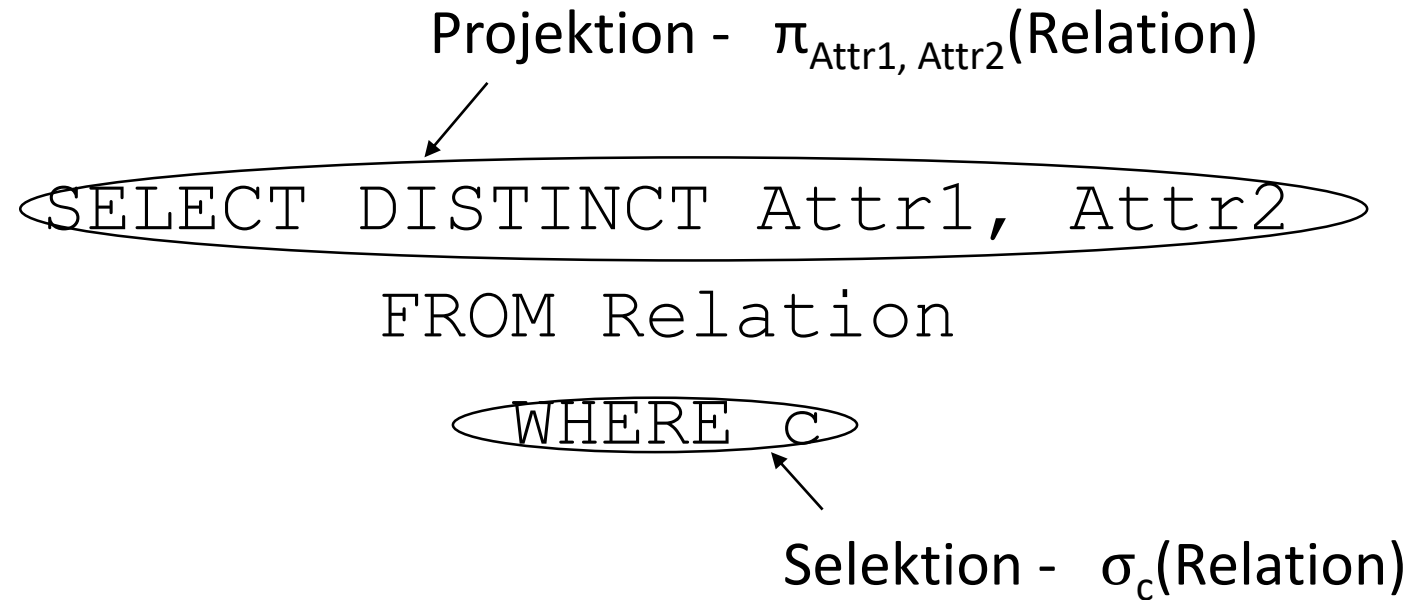
$\sigma_{\text{Name} = \text{'Schmidt'}}(\text{Studenten})$



```
SELECT DISTINCT * FROM Studenten  
WHERE Name = 'Schmidt'
```

# Aufpassen

- Nicht verwechseln:



# Zusammensetzung von Projektion und Selektion

$\pi_{\text{Name, Vorname, Ort}}(\sigma_{\text{Name} = \text{'Schmidt'}}(\text{Studenten}))$

↓

```
SELECT DISTINCT Name, Vorname, Ort  
FROM Studenten  
WHERE Name = 'Schmidt'
```

$\sigma_{\text{Name} = \text{'Schmidt'}}(\pi_{\text{Name, Vorname, Ort}}(\text{Studenten}))$

- Welches ist das äquivalente SQL Query?
- Kann man immer die Reihenfolge der Projektion und Selektion wechseln?
- Nein → die Selektion kann nach der Projektion ausgeführt werden, nur dann wenn die Selektionsbedingung nur Attribute aus der Projektion enthält

# Vereinigung, Durchschnitt, Differenz

- Vereinigung:  $R_1 \cup R_2 = \{ t \mid t \in R_1 \vee t \in R_2 \}$
- Durchschnitt:  $R_1 \cap R_2 = \{ t \mid t \in R_1 \wedge t \in R_2 \}$
- Differenz:  $R_1 - R_2 = \{ t \mid t \in R_1 \wedge t \notin R_2 \}$
- $R_1$  und  $R_2$  müssen für alle diese Operationen gleiches Relationenschema besitzen
- Wertebereiche müssen kompatibel oder vereinigungsverträglich sein
- Bem. Es gilt  $R_1 \cap R_2 = R_1 - (R_1 - R_2)$

# Vereinigung, Durchschnitt, Differenz in SQL

$R_1 \cup R_2$

```
SELECT DISTINCT *  
FROM R1
```

**UNION**

```
SELECT DISTINCT *  
FROM R2
```

$R_1 \cap R_2$

```
SELECT DISTINCT *  
FROM R1
```

**INTERSECT**

```
SELECT DISTINCT *  
FROM R2
```

$R_1 - R_2$

```
SELECT DISTINCT *  
FROM R1
```

**EXCEPT**

```
SELECT DISTINCT *  
FROM R2
```

# Kartesisches Produkt

- Das kartesische Produkt zweier Relationen  $R_1(A_1, \dots, A_n)$  und  $R_2(B_1, \dots, B_m)$  ist definiert als Relation:

$$\begin{aligned} R_1 \times R_2 = \{ t \mid & t_1 \in R_1 \wedge t_2 \in R_2 \\ & \wedge t.A_1 = t_1.A_1 \wedge \dots \wedge t.A_n = t_1.A_n \\ & \wedge t.B_1 = t_2.B_1 \wedge \dots \wedge t.B_m = t_2.B_m \} \end{aligned}$$

SQL:

```
SELECT DISTINCT *  
FROM R1, R2
```

# Kartesisches Produkt - Beispiel

A <sub>1</sub>	A <sub>2</sub>
1	A
2	B
3	C

$\times$

B <sub>1</sub>	B <sub>2</sub>
1	X
2	Y
4	Z

$=$

A <sub>1</sub>	A <sub>2</sub>	B <sub>1</sub>	B <sub>2</sub>
1	A	1	X
1	A	2	Y
1	A	4	Z
2	B	1	X
2	B	2	Y
2	B	4	Z
3	C	1	X
3	C	2	Y
3	C	4	Z



# $\theta$ -Join (Theta-Verbund)

- Auswahl bestimmter Tupel aus dem kartesischen Produkt  $R_1 \times R_2$
- Basis der Verknüpfung der Relationen: eine Bedingung  $c$

$$R_1 \bowtie_c R_2 = \sigma_c(R_1 \times R_2)$$

Bsp.

Studenten  $\bowtie_{\text{Studenten.MatrikelNr} = \text{Enrolled.MatrikelNr}}$  Enrolled

SQL:

```
SELECT DISTINCT *  
FROM Studenten, Enrolled  
WHERE Studenten.MatrikelNr  
= Enrolled.MatrikelNr
```

**oder**

```
SELECT DISTINCT *  
FROM Studenten  
INNER JOIN Enrolled ON  
Studenten.MatrikelNr =  
Enrolled.MatrikelNr
```

# Equi-Join

- Einen  $\theta$ -Join der Form  $R_1 \bowtie_{R_1.A_i = R_2.B_j} R_2$  nennt man Equi-Join
- Notation für Equi-Join um zu unterscheiden:  $R_1 \bowtie_{E(R_1.A_i = R_2.B_j)} R_2$
- Die Bedingung muss der Form einer Gleichwertigkeit zwischen Attribute der ersten und der zweiten Relation sein
- Das Ergebnis enthält nur einen der Attribute, da es redundant ist beide zu behalten (die Attribute sind gleich)

# Equi-Join Beispiel

Kurse

KursId	Titel
Alg1	Algorithmen1
DB1	Datenbanken1
DB2	Datenbanken2

Enrolled

MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10

Kurse  $\bowtie_{E(Kurse.KursId=Enrolled.KursId)}$  Enrolled

KursId	Titel	MatrNr	Note
Alg1	Algorithmen1	1234	7
Alg1	Algorithmen1	1235	8
DB1	Datenbanken1	1234	9
DB2	Datenbanken2	1234	7
DB1	Datenbanken1	1236	10

# Natürlicher Verbund

- Verknüpft zwei Relationen indem alle gleichbenannten Attribute der beiden Relationen betrachtet werden und nur einen der gleichen Attribute kommt in das Ergebnis vor (ohne Redundanzen)
- Qualifizierende Tupel müssen für diese gleichbenannten Attribute gleiche Werte aufweisen, um in das Ergebnis einzugehen
- Gibt es kein gemeinsames Attribut so ist das Ergebnis das kartesische Produkt

Kurse

KursId	Titel
Alg1	Algorithmen1
DB1	Datenbanken1
DB2	Datenbanken2

Enrolled

MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10

Kurse  $\bowtie$  Enrolled

KursId	Titel	MatrNr	Note
Alg1	Algorithmen1	1234	7
Alg1	Algorithmen1	1235	8
DB1	Datenbanken1	1234	9
DB2	Datenbanken2	1234	7
DB1	Datenbanken1	1236	10

# Division

- Die Relation  $R_1$  enthält Attribute X und Y und  $R_2$  enthält den Attribut Y.

$$R_1 \div R_2 = \{ \langle X \rangle \mid \forall \langle Y \rangle \in R_2 : \exists \langle X, Y \rangle \in R_1 \}$$

- $R_1 \div R_2$  (oder  $R_1 / R_2$ ) enthält alle X Tupeln so dass für jedes Y Tupel in  $R_2$ , ein XY Tupel in  $R_1$  existiert
- X und Y können auch Mengen von Attributen sein

# Division

- Nicht als primitiver Operator, aber nützlich
- Die Division wird dann eingesetzt, wenn die Frage „**für alle**“ enthält
- Beispielfragestellungen für eine Division:
  - Welche Personen haben eine Kundenkarte von **allen** Filialen?
  - Welche Mitarbeiter arbeiten an **allen** Projekten?
  - Welche Studenten hören **alle** Vorlesungen von Prof. X?

# Division

- Darstellung des Quotienten durch die Basisoperatoren:
  - Idee: Berechne alle X Werte, die von irgendeinem Y Wert aus  $R_2$  disqualifiziert wird
  - X wird disqualifiziert wenn für einen Y der Tupel XY nicht in  $R_1$  enthalten ist:
$$\pi_X( ( \pi_X(R_1) \times R_2 ) - R_1 )$$
  - Der Quotient  $R_1 \div R_2$  enthält dann alle X Werte aus  $R_1$ , die nicht disqualifiziert sind:

$$R_1 \div R_2 = \pi_X(R_1) - \pi_X( ( \pi_X(R_1) \times R_2 ) - R_1 )$$

# Division - Beispiel

$R_1$

<b>A</b>	<b>B</b>
4	3
4	1
4	7
8	3
8	1
8	7

$R_2$

<b>A</b>
4
8

$R_1 \div R_2$

<b>B</b>
3
1
7



# Umbenennen von Relationen und Attributen

- Umbenennung unterscheidet sich von den anderen Operatoren dadurch, dass keine Berechnung vorgenommen wird
- Operator ist aber notwendig, wenn eine Relation mehrfach in einer Anfrage vorkommt (z.B. Join)
- $\rho_S(R)$  : Relation R wird in Relation S umbenannt
- $\rho_{B \leftarrow A}(R)$  : Attribut A der Relation R wird umbenannt in B
- Das Relationenschema wird nicht geändert (nur eventuell Namen von Attributen)

# Zuweisungsoperation

- Die Zuweisungsoperation  $\leftarrow$  ist eine Methode komplexe Abfragen zu representieren
- Eine Abfrage kann in einer temporären Variable gespeichert werden

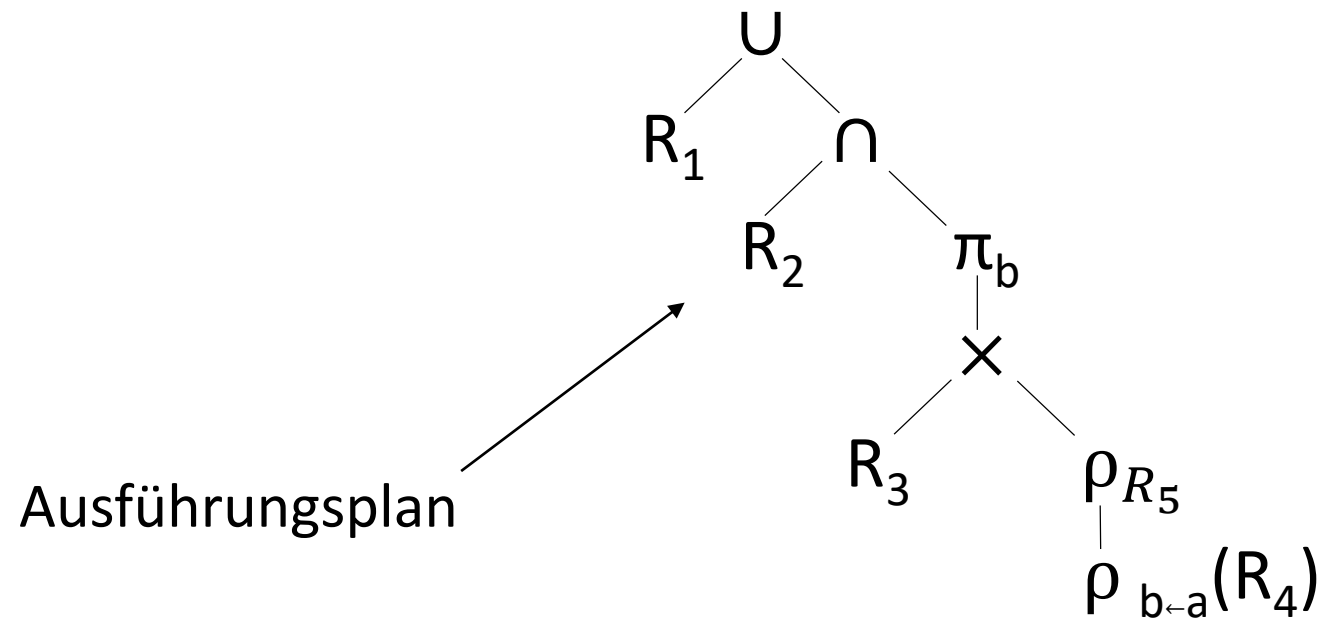
$$\text{Temp} \leftarrow \pi_x(R_1 \times R_2)$$

- Dann kann man diese Variable in weiteren Abfragen benutzen

$$\text{Erg} \leftarrow \text{Temp} - R_3$$

# Komplexe Abfragen

$$R_1 \cup (R_2 \cap \pi_b (R_3 \times \rho_{R_5}(\rho_{b \leftarrow a}(R_4))))$$



Studenten

MatrNr	Name	Vorname
1234	Schmidt	Hans
1235	Meisel	Amelie
1236	Krause	Julia
1237	Rasch	Lara
1238	Schmidt	Christian

Kurse

KursId	Titel	ECTS
Alg1	Algorithmen1	6
DB1	Datenbanken1	6
DB2	Datenbanken2	5

Enrolled

MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10

Geben Sie die Namen der Studenten aus, die für den Kurs `BD1` angemeldet sind

- Lsg1.

$$\pi_{\text{Name}}((\sigma_{\text{KursId}=\text{'BD1'}}(\text{Enrolled})) \bowtie \text{Studenten})$$

- Lsg2.

$$\rho_{\text{Temp1}}(\sigma_{\text{KursId}=\text{'BD1'}}(\text{Enrolled}))$$

$$\rho_{\text{Temp2}}(\text{Temp1} \bowtie \text{Studenten})$$

$$\pi_{\text{Name}}(\text{Temp2})$$

- Lsg3.

$$\pi_{\text{Name}}(\sigma_{\text{KursId}=\text{'BD1'}}(\text{Enrolled} \bowtie \text{Studenten}))$$

Geben Sie die Namen der Studenten aus, die für einen Kurs mit 5 ECTS angemeldet sind

- Lsg1.

$$\pi_{\text{Name}}((\sigma_{\text{ECTS}=5}(\text{Kurse})) \bowtie \text{Enrolled} \bowtie \text{Studenten})$$

- Lsg2.

$$\pi_{\text{Name}}(\pi_{\text{MatrNr}}(\pi_{\text{KursId}}(\sigma_{\text{ECTS}=5}(\text{Kurse})) \bowtie \text{Enrolled}) \bowtie \text{Studenten})$$

- Lsg2 ist effizienter. Ein Abfrageoptimierer würde, gegeben die erste Abfrage, die zweite Abfrage finden.

Geben Sie die Namen der Studenten aus, die für einen Kurs mit 5 **oder** 6 ECTS angemeldet sind

- Wir können erstmal die Kurse mit 5 oder 6 ECTS ausgeben und dann die Studenten die in einem dieser Kurse angemeldet sind

$$\rho_{\text{TempKurse}}(\sigma_{\text{ECTS}=5 \vee \text{ECTS}=6}(\text{Kurse}))$$
$$\pi_{\text{Name}}(\text{TempKurse} \bowtie \text{Enrolled} \bowtie \text{Studenten})$$

- Was passiert wenn wir „oder“ mit „und“ ersetzen

Geben Sie die Namen der Studenten aus, die für einen Kurs mit 5 ECTS **und** einen Kurs mit 6 ECTS angemeldet sind

- Die vorige Idee funktioniert nicht mehr.
- Wir müssen die Studenten finden, die in einem 5 ECTS Kurs angemeldet sind und die die in einem 6 ECTS Kurs angemeldet sind und den Durchschnitt berechnen

$$\rho_{\text{Temp5}}(\pi_{\text{MatrNr}}(\sigma_{\text{ECTS}=5}(\text{Kurse}) \bowtie \text{Enrolled}))$$
$$\rho_{\text{Temp6}}(\pi_{\text{MatrNr}}(\sigma_{\text{ECTS}=6}(\text{Kurse}) \bowtie \text{Enrolled}))$$
$$\pi_{\text{Name}}((\text{Temp5} \cap \text{Temp6}) \bowtie \text{Studenten})$$



Geben Sie die Namen der Studenten aus, die **für alle** Kurse angemeldet sind

- „**Für alle**“ → wir benutzen Division

$$\rho_{\text{TempMatrNr}}(\pi_{\text{MatrNr,KursId}}(\text{Enrolled}) / \pi_{\text{KursId}}(\text{Kurse})) \\ \pi_{\text{Name}}(\text{TempMatrNr} \bowtie \text{Studenten})$$

# Erweiterte Relationale Algebra Operatoren

- Erweiterte Projektion
- Aggregat Funktionen
- Outer Join
- Datenbank Änderungen

# Erweiterte Projektion

- Erweitert die Projektion, indem arithmetische Funktionen als Projektionsbedingung benutzt werden können

$$\pi_{F_1, \dots, F_n}(R)$$

- $F_1, \dots, F_n$  sind arithmetische Funktionen, die Konstante oder Attribute der Relation  $R$  enthalten

# Aggregat Funktionen

- Haben mehrere Werte als Input und ein Wert als Output:
  - avg: Mittelwert
  - min: Minimum der Werte
  - max: Maximum der Werte
  - sum: Summe der Werte
  - count: Anzahl der Werte

# Aggregat Funktionen in Relationale Algebra

$$G_1, G_2, \dots, G_n \vartheta_{F_1(A_1), F_2(A_2), \dots, F_n(A_n)}(R)$$

- $G_1, G_2, \dots, G_n$  – eine Liste von Attributen worauf wir gruppieren wollen
- $F_i$  – Aggregatfunktion
- $A_i$  – Name eines Attributes

# Aggregat Funktionen - Beispiel

Relation R:

<b>A</b>	<b>B</b>	<b>C</b>
a	2	5
b	3	3
a	4	4

$$\vartheta_{\text{sum}(C)}(R) \Rightarrow 12$$

# Outer Join

- Erweiterung von Join-Operationen:
  - **Left Outer Join**  $\bowtie$  - alle Tupel aus der linken Relation, die keinen Join-Partner in der rechten Relation haben, werden trotzdem ausgegeben
  - **Right Outer Join**  $\bowtie$  - alle Tupel aus der rechten Relation, die keinen Join-Partner in der linken Relation haben, werden trotzdem ausgegeben
  - **Full Outer Join**  $\bowtie$  - alle Tupel sowohl der linken als auch der rechten Relation, die keinen Join-Partner haben, werden trotzdem ausgegeben
- Null-Werte werden benutzt:
  - Tupeln aus der Relation R, die keinen Join-Partner in der Relation S hatten enthalten Null-Werte für die entsprechenden Spalten der Relation S
  - Ein Null-Wert heißt unbekannt oder inexistent
  - Alle Vergleiche mit einem Null-Wert werden in der Regel als FALSE bewertet

# Outer Join - SQL

- **RIGHT JOIN** (alternativ **RIGHT OUTER JOIN**)

```
SELECT *  
FROM Studenten RIGHT JOIN Studiengang  
ON Studenten.SgNr = Studiengaenge.SgNr
```

- **LEFT JOIN** (alternativ **LEFT OUTER JOIN**)

```
SELECT *  
FROM Studiengaenge LEFT JOIN Studenten  
ON Studenten.SgNr = Studiengaenge.SgNr
```

- **FULL OUTER JOIN**

- Nicht in allen DB-Systemen verfügbar (z.B. MySQL nicht)



# Datenbank Änderungen

- Der Inhalt der Datenbank kann durch folgenden Operationen geändert werden:
  - Löschen:  $R \leftarrow R - E$
  - Einfügen:  $R \leftarrow R \cup E$
  - Aktualisierung/Updating:  $R \leftarrow \pi_{F_1, \dots, F_n}(R)$