

SQL-Übung

Wintersemester 03/04

Übungsleiter: Dipl.-Inform. Tom Gelhausen



Grundoperationen

σ	Selektion	relation \times bedingung \mapsto relation
π	Projektion	relation \times attributfolge \mapsto relation
\times	Kartesisches Prod.	relation \times relation \mapsto relation
\cup	Vereinigung	relation \times relation \mapsto relation
\setminus	Differenz	relation \times relation \mapsto relation

Makrooperationen

\cap	Durchschnitt	relation \times relation \mapsto relation
\bowtie_{θ}	Theta-Verbindung	relation \times bedingung \times relation \mapsto relation
\bowtie	nat. Verbindung	relation \times relation \mapsto relation
(\div)	Division	relation \times relation \mapsto relation)



- Grundlage der Verbindungsoperationen in der relationalen Algebra ist das kartesische Produkt: $\rho_1 \bowtie_{\Theta} \rho_2 = \sigma_{\Theta}(\rho_1 \times \rho_2)$
- Kartesisches Produkt ($\rho_1 \times \rho_2$) in SQL:

```
SELECT *  
FROM Rel1, Rel2, Rel3;
```

- Die SELECT-Klausel „*“
 - entspricht der Projektionsfunktion $\pi(\rho) := \rho$,
 - es werden also alle Tupelkomponenten zurückgegeben
 - in der Reihenfolge, in der sie in *Rel1* ◦ *Rel2* ◦ *Rel3* stehen
 - mit den Namen mit denen sie in *Rel1* ◦ *Rel2* ◦ *Rel3* stehen (bei Namensgleichheit durch *Rel.-name* ◦ „.“ ◦ *Attrib.-name* qualifiziert)
- Die FROM-Klausel enthält eine Liste der Relationen, aus denen das kartesische Produkt gebildet werden soll
- Die WHERE-Klausel ist implizit wahr und darf als optionaler Bestandteil weggelassen werden.



- Grundlage der Verbindungsoperationen in der relationalen Algebra ist das kartesische Produkt: $\rho_1 \bowtie_{\Theta} \rho_2 = \sigma_{\Theta}(\rho_1 \times \rho_2)$
- Selektion bezüglich eines Prädikates Θ ($\sigma_{\Theta}(\rho)$) in SQL:

```
SELECT *  
FROM Rel  
WHERE  $\Theta(t)$ ;
```

- Das Prädikat Θ wird auf jedes Tupel aus der Relation `Rel` angewendet: Wenn es zu Wahr ausgewertet wird ist das Tupel in der Ergebnismenge, sonst nicht.
- Θ kann ein atomares Prädikat sein:
 - atomaren Prädikate in SQL: `=`, `>`, `<`, `>=`, `<=`, und weitere
 - jedoch Zugriff nur über Selektoren (auf Attribute des akt. Tupels)
⇒ nur Operatoren auf atomaren Typen

Beispiele: `Name='Klaus' oder Beruf='Staplerfahrer'`



- Grundlage der Verbindungsoperationen in der relationalen Algebra ist das kartesische Produkt: $\rho_1 \bowtie_{\Theta} \rho_2 = \sigma_{\Theta}(\rho_1 \times \rho_2)$
- Selektion bezüglich eines Prädikates Θ ($\sigma_{\Theta}(\rho)$) in SQL:

```
SELECT *  
FROM Rel  
WHERE  $\Theta(t)$ ;
```

- Das Prädikat Θ wird auf jedes Tupel aus der Relation `Rel` angewendet: Wenn es zu Wahr ausgewertet wird ist das Tupel in der Ergebnismenge, sonst nicht.
- Θ kann ein zusammengesetztes Prädikat sein
 - SQL kennt hierfür die Schlüsselwörter: AND, OR, NOT, ...
 - Für die atomaren Operanden (=atomare Prädikate) gilt, dass alle Selektoren einer WHERE-Klausel immer auf das gleiche (=aktuelle) Tupel verweisen!

Beispiel: `Name='Klaus' AND Beruf='Staplerfahrer'`



- Grundlage der Verbindungsoperationen in der relationalen Algebra ist das kartesische Produkt: $\rho_1 \bowtie_{\Theta} \rho_2 = \sigma_{\Theta}(\rho_1 \times \rho_2)$
- Frage: Wie passen folgende zwei Konzepte zusammen?



- Betrachte beispielsweise:

Rel1 := { (a,a),(b,b) }

Rel2 := { (2, "X"), (3, "Y") }

Rel3 := { (63,a,l),(15,h,p) }

Rel := { (a,a,2, "X", 63,a,l),

(a,a,2, "X", 15,h,p),

(a,a,3, "Y", 63,a,l),

(a,a,3, "Y", 15,h,p),

(b,b,2, "X", 63,a,l),

(b,b,2, "X", 15,h,p),

(b,b,3, "Y", 63,a,l),

(b,b,3, "Y", 15,h,p) }

⇒ Rel = Rel1 × Rel2 × Rel3



- Grundlage der Verbindungsoperationen in der relationalen Algebra ist das kartesische Produkt: $\rho_1 \bowtie_{\Theta} \rho_2 = \sigma_{\Theta}(\rho_1 \times \rho_2)$
- Entsprechend wird der Theta-Join in SQL wie folgt ausgedrückt:

```
 $\rho_1 \bowtie_{\Theta} \rho_2 =$  SELECT *  
FROM (SELECT *  
FROM  $\rho_1, \rho_2$  )  
WHERE  $\Theta$ ;
```

Die gebräuchlichere Form ist allerdings:

```
SELECT *  
FROM  $\rho_1, \rho_2$   
WHERE  $\Theta$ ;
```

Feststellungen:

- Das Ergebnis einer Anfrage ist selbst wieder eine gültige Relation.
- Die Ergebnisrelation ist allerdings nur temporär und wird nach der Ergebnisverarbeitung wieder verworfen.
- Diese Eigenschaft ermöglicht so genannte „**geschachtelte Anfragen**“.



- Zu welcher Kategorie gehört der Artikel mit der Artikelnummer 4711?

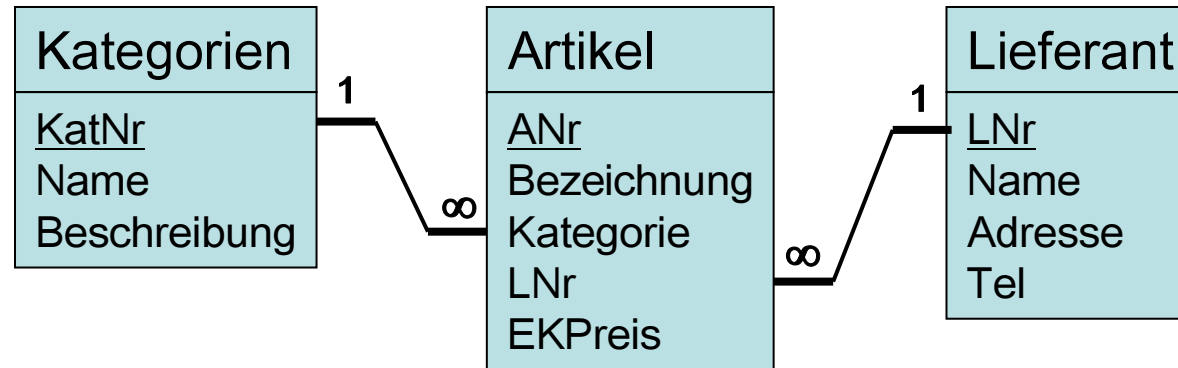
```
SELECT k.Name
FROM Artikel a, Kategorien k
WHERE a.Kategorie=k.KatNr
AND a.ANr=4711;
```

- Aber woher weiß man, dass es „Bestellungen“, „VersandUeber“, „BestellNr“, etc. heißt? Und dass man „VersandUeber“ und „FirmenNr“ vergleichen muss?

⇒ Schema!



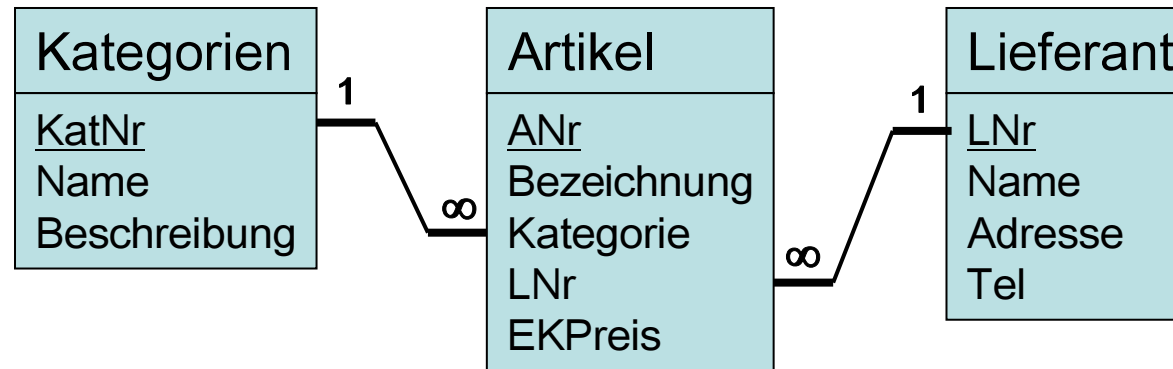
- Schema:



- Relationen werden als Kasten dargestellt
- Attribute, die miteinander verbunden werden „könnten“ sind durch Linien miteinander verbunden
- Was bedeuten „1“ und „∞“ an diesen Linien?



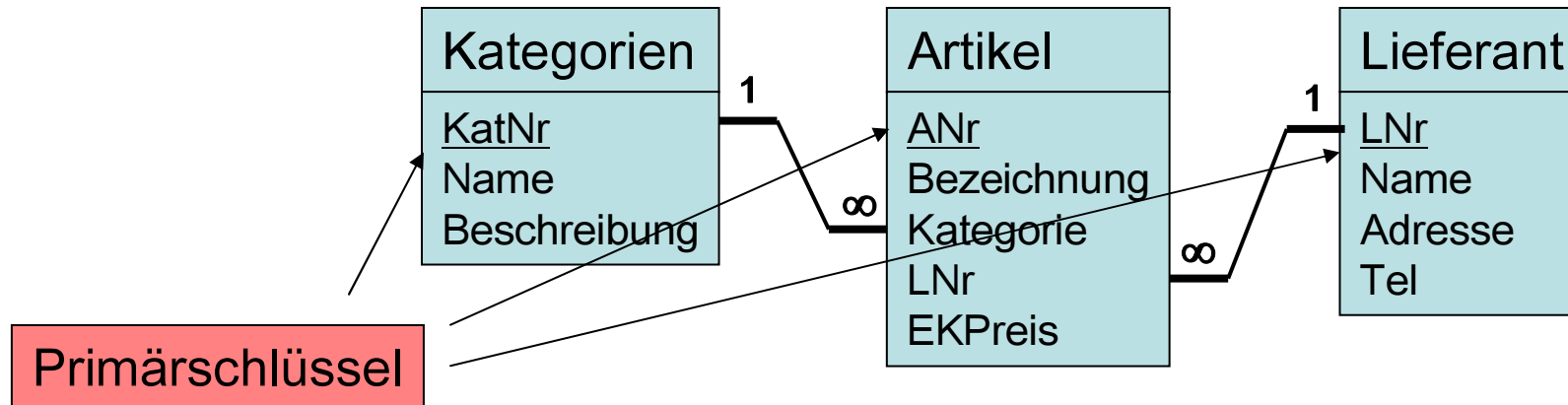
- Schema:



- Dem DBMS vorgegebene Konsistenzbedingung, die die Kopplung zweier Relationen ρ_1 und ρ_2 ermöglicht, so dass
 - die in ρ_1 unter der Attributfolge fk auftretenden Werte
 - in ρ_2 unter der Attributfolge pk auftreten,also: $\pi_{fk}(\rho_1) \subseteq \pi_{pk}(\rho_2)$
- Es besteht **referentielle Konsistenz** von $\rho_1.fk$ nach $\rho_2.pk$.
- Ist pk Schlüssel von ρ_2 , dann nennt man fk **Fremdschlüssel** (**foreign key**) in ρ_1 .



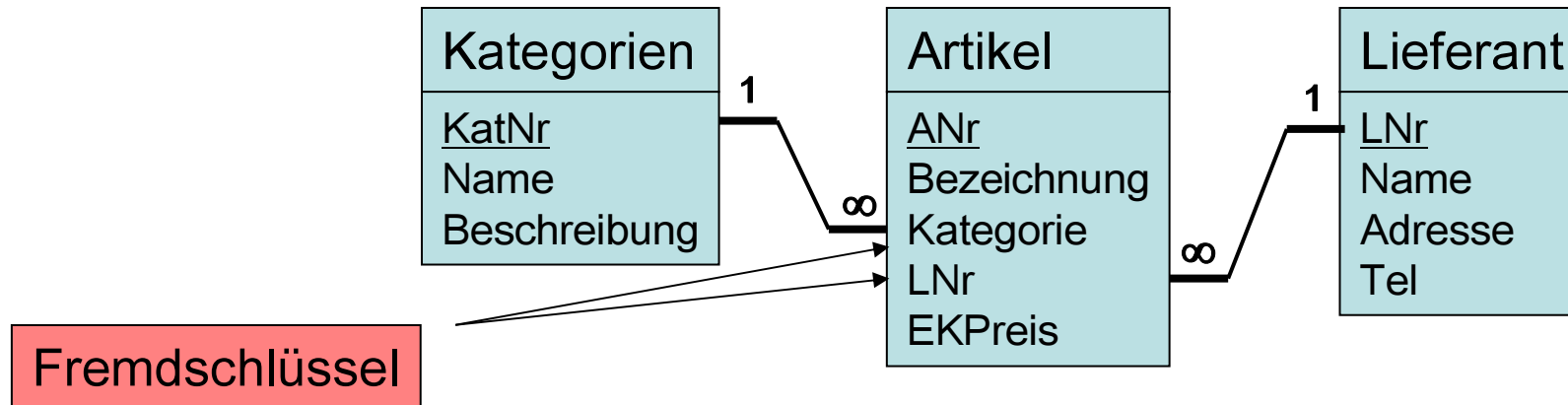
- Schema:



- „KatNr“ ist Primärschlüssel in „Kategorien“ und charakterisiert alle Kategorien eindeutig. Jede „KatNr“ gibt es in der Relation „Kategorien“ nur ein mal.
- „ANr“ ist Primärschlüssel in „Artikel“ und charakterisiert alle Artikel eindeutig. Jede „ANr“ gibt es in der Relation „Artikel“ nur ein mal.
- „LNr“ ist Primärschlüssel in „Lieferanten“ und charakterisiert alle Lieferanten eindeutig. Jede „LNr“ gibt es in der Relation „Lieferanten“ nur ein mal.

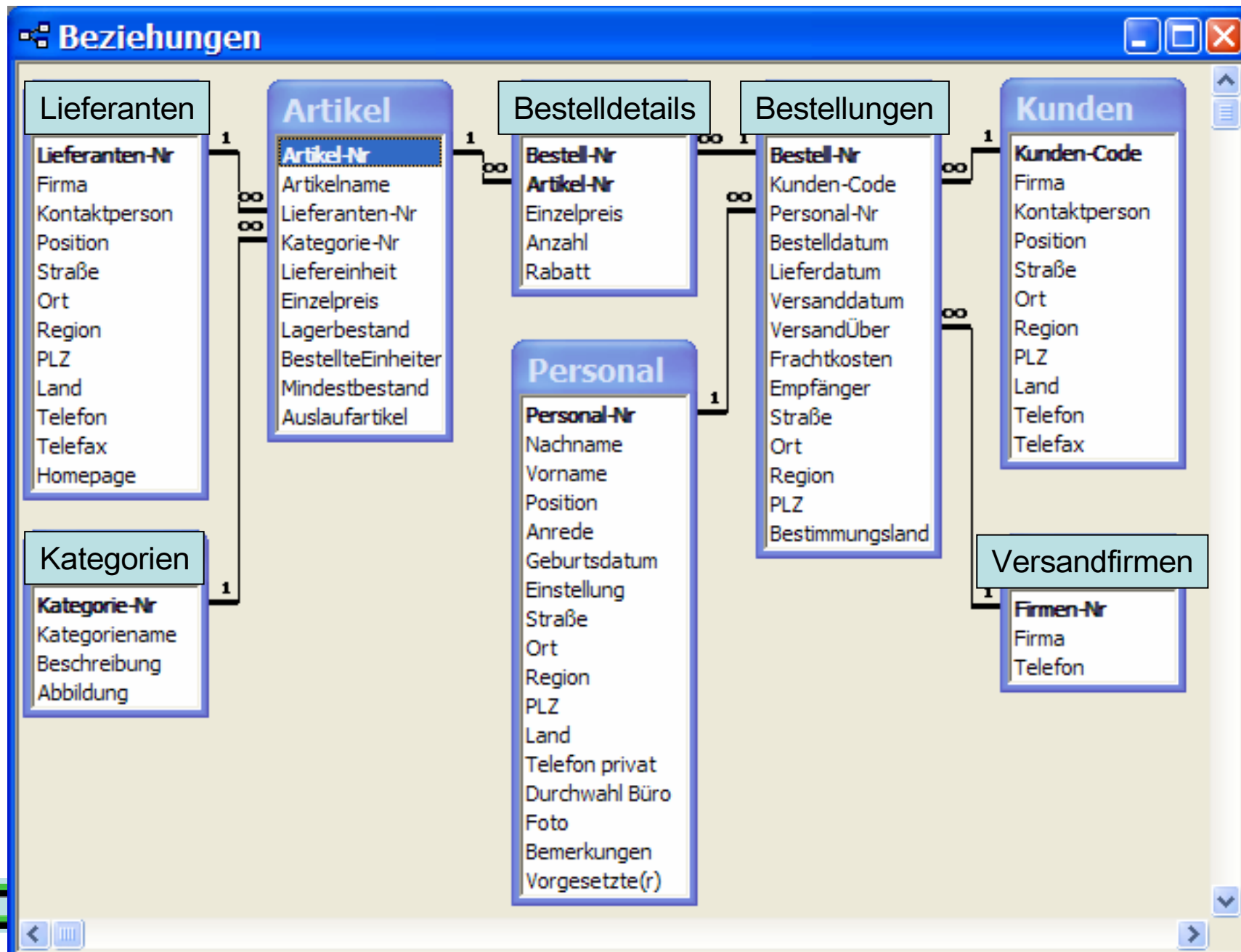


- Schema:

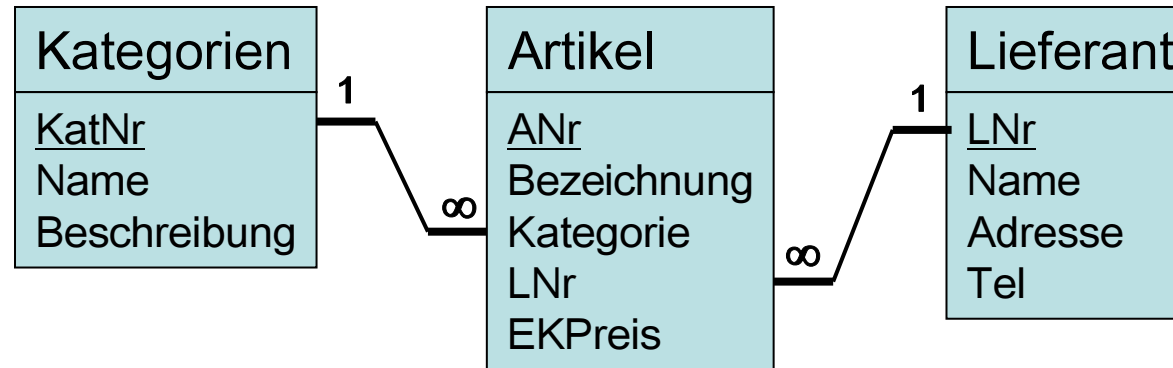


- „Kategorie“ ist Fremdschlüssel in „Artikel“ und legt somit fest, dass in dieser Spalte nur Werte stehen dürfen, die auch in der Spalte „KNr“ der Relation „Kategorien“ vorkommen. In der Spalte „Kategorie“ der Relation „Artikel“ dürfen die Werte aber beliebig oft vorkommen.
- „LNr“ ist Fremdschlüssel in „Artikel“ und legt somit fest, dass in dieser Spalte nur Werte stehen dürfen, die auch in der Spalte „LNr“ der Relation „Lieferant“ vorkommen. In der Spalte „LNr“ der Relation „Artikel“ dürfen die Werte aber beliebig oft vorkommen.





- Schema:



```

SELECT *
FROM Artikel a, Kategorien k
WHERE a.Kategorie=k.KatNr;
    
```

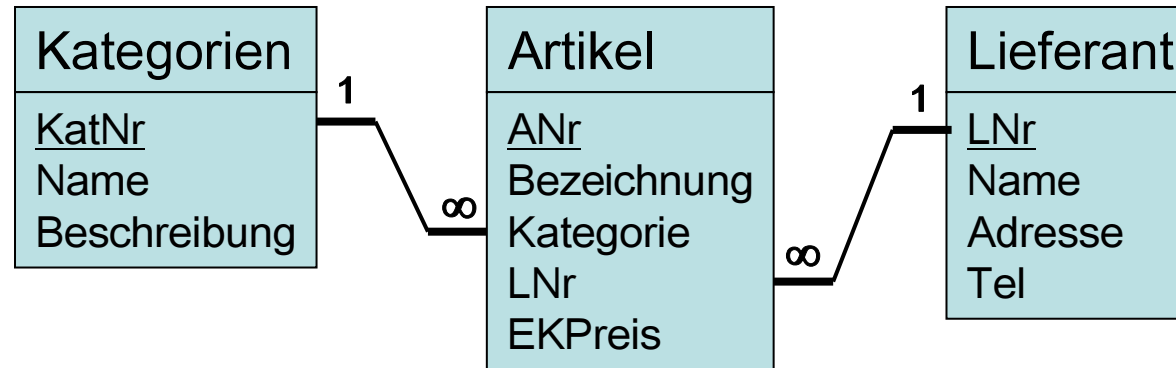
mit $\rho_1 = \text{Artikel}$, $\rho_2 = \text{Kategorien}$, $\Theta = \underbrace{a.Kategorie=k.KatNr}$

„Equi-Join“

$\sigma_{\text{Kategorie=KatNr}}(\text{Artikel} \times \text{Kategorie})$



- Schema:

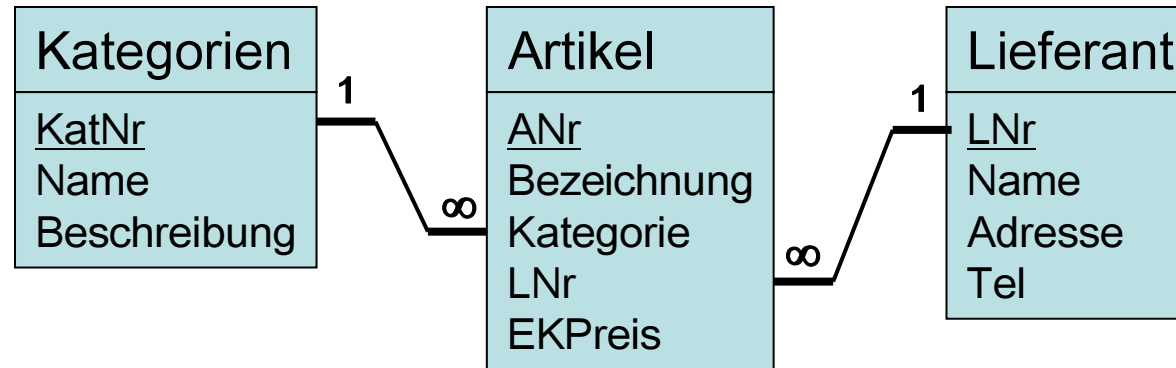


```
SELECT *
FROM Artikel a
      INNER JOIN Kategorien k
      ON a.Kategorie=k.KatNr
[WHERE ...];
```

ebenfalls: $\sigma_{\text{Kategorie}=\text{KatNr}}(\text{Artikel} \times \text{Kategorie})$



▪ Schema:



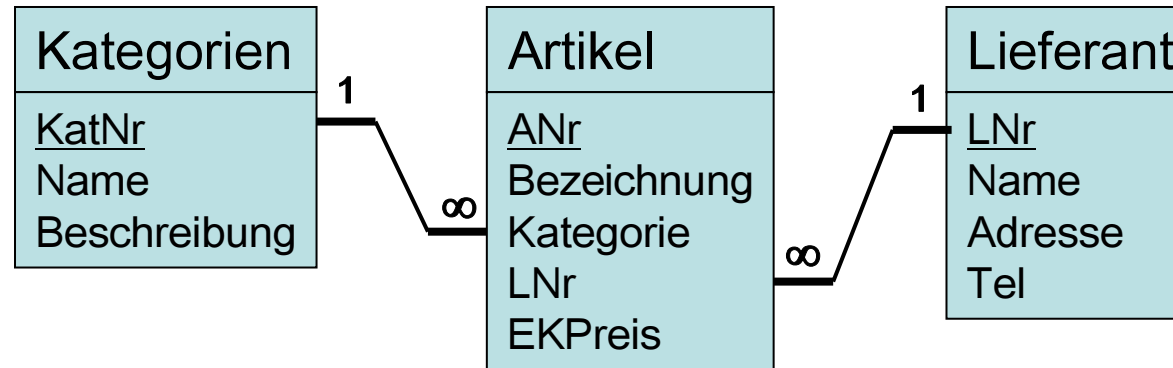
```
SELECT  a.Bezeichnung, l.Name
FROM    Artikel a, Lieferant l
WHERE   a.LNr=l.LNr;
```

Liefert eine Liste aus Zeichenkettenpaaren:

- Die erste Komponente enthält die Bezeichnung eines Artikels
- Die zweite Komponente enthält den Namen des Lieferanten des Artikels.



▪ Schema:



```

SELECT a.Bezeichnung, k.Name, l.Name
FROM Artikel a
      INNER JOIN Lieferant l ON a.LNr=l.LNr
      INNER JOIN Kategorie k
      ON a.Kategorie=k.KNr
WHERE ANr=4711;
    
```

$\Pi_{a.Bezeichnung, k.Name, l.Name} \sigma_{ANr=4711} (((\text{Artikel} \bowtie \text{Lieferant}) \bowtie_{Kategorie=KNr} \text{Kategorie}))$



- Exemplarische Ausführung:

Kategorien		
KatNr	Name	Beschreibung
1	Getränke	Alkoholfreie Getränke, Kaf...
2	Gewürze	Süße und saure Soßen, Ge...
3	Süßwaren	Desserts und Süßigkeiten
4	Milchprodukte	Käsesorten
5	Getreideprodukte	Brot, Kräcker, Nudeln, Müsli
6	Fleischprodukte	Fleisch-Fertiggerichte
7	Naturprodukte	Getrocknete Früchte, Tofu...
8	Meeresfrüchte	Meerespflanzen und -früch...

Lieferanten		
LNr	Name	Tel
1	Exotic Liquids	(71) 555-2222
2	New Orleans Cajun	(100) 555-4822
3	Grandma Kelly's	(313) 555-5735
4	Tokyo Traders	(03) 3555-5011
5	Las Cabras	(98) 598 76 54
6	Mayumi's	(06) 431-7877
7	Pavlova, Ltd.	(03) 444-2343
8	Specialty Biscuits	(26) 555-4448
9	PB Knäckebröd AB	031-987 65 43
10	Refrescos Americanas	(11) 555 4640
11	Heli Süßwaren	(010) 9984510
12	Plutzer Lebensmittel	(069) 992755
13	Nord-Ost-Fisch	(04721) 8713
14	Formaggi Fortini	(0544) 60323
15	Norske Meierier	(0)2-953010
16	Bigfoot Breweries	(503) 555-9931
17	Svensk Sjöföda AB	08-123 45 67
18	Joyeux ecclésiastiques	(1) 03.83.00.68
19	New England Seafood	(617) 555-3267

Artikel				
ANr	Bezeichnung	LNr	KatNr	EKPreis
1	Chai	1	1	18,00 €
2	Chang	1	1	19,00 €
3	Aniseed Syrup	1	2	10,00 €
4	Chef Anton's Cajun Seasoning	2	2	22,00 €
5	Chef Anton's Gumbo Mix	2	2	21,35 €
6	Grandma's Boysenberry Spread	3	2	25,00 €
7	Uncle Bob's Organic Dried Pears	3	7	30,00 €
8	Northwoods Cranberry Sauce	3	2	40,00 €

```
SELECT a.Bezeichnung, k.Name AS Kategorie, l.Name AS Firma
FROM Artikel a
INNER JOIN Lieferant l ON a.LNr=l.LNr
INNER JOIN Kategorie k ON a.Kategorie=k.KNr;
```

15	Genen Shouyu	6	2	15,50 €
...				



- Exemplarische Ausführung:

Kategorien		
KatNr	Name	Beschreibung
1	Getränke	
2	Gewürze	
3	Süßwaren	
4	Milchprodukte	
5	Getreide	
6	Fleischprodukte	
7	Naturprodukte	
8	Meeresfrüchte	

Lieferanten		
LNr	Name	Tel
1	Exotic Liquids	555-2222
2	Exotic Liquids	555-4822
3	Exotic Liquids	555-5735
4	Exotic Liquids	3555-5011
5	Exotic Liquids	598 76 54
6	New Orleans Cajun	431-7877
7	New Orleans Cajun	444-2343
8	Grandma Kelly's	555-4448
9	Grandma Kelly's	987 65 43
10	Mayumi's	555 4640
11	Las Cabras	9984510
12	Las Cabras	992755
13	Tokyo Traders	721) 8713
14	Grandma Kelly's	44) 60323
15	Grandma Kelly's	2-953010
16	Mayumi's	3) 555-9931
17	Tokyo Traders	123 45 67
18	Mayumi's	03.83.00.68
19	Mayumi's	7) 555-3267

Artikel	
ANr	Bezeichnung
1	Chai
2	Chang
3	Aniseed Syrup
4	Chef Anton's Cajun Seasoning
5	Chef Anton's Gumbo Mix
6	Grandma's Boysenberry Spread
7	Uncle Bob's Organic Dried Pears
8	Northwoods Cranberry Sauce

Bezeichnung	Kategorie	Firma	Tel
Chai	Getränke	Exotic Liquids	555-2222
Chang	Getränke	Exotic Liquids	555-4822
Aniseed Syrup	Gewürze	Exotic Liquids	555-5735
Chef Anton's Cajun Seasoning	Gewürze	New Orleans Cajun	3555-5011
Chef Anton's Gumbo Mix	Gewürze	New Orleans Cajun	598 76 54
Grandma's Boysenberry Spread	Gewürze	Grandma Kelly's	431-7877
Northwoods Cranberry Sauce	Gewürze	Grandma Kelly's	444-2343
Genen Shouyu	Gewürze	Mayumi's	555-4448
Queso Cabrales	Milchprodukte	Las Cabras	987 65 43
Queso Manchego La Pastora	Milchprodukte	Las Cabras	555 4640
Mishi Kobe Niku	Fleischprodukte	Tokyo Traders	9984510
Uncle Bob's Organic Dried Pears	Naturprodukte	Grandma Kelly's	992755
Tofu	Naturprodukte	Mayumi's	721) 8713
Ikura	Meeresfrüchte	Tokyo Traders	44) 60323
Konbu	Meeresfrüchte	Mayumi's	2-953010
...			3) 555-9931

```
SELECT a.Bezeichnung, k.Name AS Kategorie, l.Name AS Firma
FROM Artikel a
INNER JOIN Lieferant l ON a.LNr=l.LNr
INNER JOIN Kategorie k ON a.Kategorie=k.KNr;
```

15	Genen Shouyu	6	2	15,50 €
...				



- Vereinigung: Der „UNION“-Operator verbindet einzelne `SELECT`-Anfragen von attributgleichem Typ und typkompatiblem Ergebnis

Beispiel: $Rel1 := \{ (1,1), (2,2) \}$, $Rel2 := \{ (2,2), (3,3) \}$

```
SELECT *  
FROM Rel1
```

UNION

```
SELECT *  
FROM Rel2
```

Ergebnis: $\{ (1,1), (2,2), (3,3) \}$ (Duplikate wurden entfernt)

```
SELECT *  
FROM Rel1
```

UNION ALL

```
SELECT *  
FROM Rel2
```

Ergebnis: $\{ (1,1), (2,2), (2,2), (3,3) \}$ (Duplikate wurden nicht entfernt)



- Gleiche Nutzungsbedingungen wie für UNION:
 - Durchschnitt: „INTERSECT“-Operator
 - Differenzbildung: „EXCEPT“-Operator
- **Hinweis:** SQL fass Relationen als Vielfachmengen und nicht als streng mathematische Mengen auf.
 - Duplikate werden also sowohl in den Quellrelationen, als auch in den Ergebnisrelationen zugelassen
 - Duplikate werden von den (meisten) SQL-Befehlen nicht automatisch beseitigt
 - Abhilfe mit „DISTINCT“:

Beispiel: Sei $Rel := \{ (1,1), (1,2), (1,3) \}$, Schema: (a:int, b:int)

```
SELECT a  
FROM Rel;
```

Ergebnis: $\{ (1), (1), (1) \}$

```
SELECT DISTINCT a  
FROM Rel;
```

Ergebnis: $\{ (1) \}$



- Welche Firma hat wie viel Frachtkosten bei uns verdient?
- Erster Ansatz:

Northwind: SQL: Query Results

Firma	Frachtkosten
Federal Shipping	\$32.38
Speedy Express	\$11.61
United Package	\$65.83
Speedy Express	\$41.34
United Package	\$51.30
United Package	\$58.17
United Package	\$22.98
Federal Shipping	\$148.33
United Package	\$13.97
Federal Shipping	\$81.91
Speedy Express	\$140.51
Federal Shipping	\$3.25
Speedy Express	\$5,509.00
United Package	\$305.00
Federal Shipping	\$48.29
Federal Shipping	\$14,606.00
Federal Shipping	\$3.67
Speedy Express	\$55.28
Federal Shipping	\$25.73
Speedy Express	\$208.58
Federal Shipping	\$66.29
Speedy Express	\$4.56
Speedy Express	\$11.61

The screenshot shows the phpPgAdmin interface with the following elements:

- Window title: phpPgAdmin - SQL - Mozilla Fir...
- Navigation tabs: SQL (selected), Find
- Section header: SQL
- Database dropdown: Northwind
- SQL query text:

```
SELECT v."Firma", b."Frachtkosten"  
FROM "Bestellungen" b, "versandfirmen" v  
WHERE b."Versandueber"=v."FirmenNr";
```
- Buttons: Go, Explain, Reset

- Welche Firma hat wie viel Frachtkosten bei uns verdient?
- Zweiter Ansatz:

Northwind: SQL: Query Results

Firma	Frachtkosten
Federal Shipping	\$32.38
Federal Shipping	\$148.33
Federal Shipping	\$81.91
Federal Shipping	\$3.25
Federal Shipping	\$48.29
Federal Shipping	\$14,606.00
Federal Shipping	\$3.67
Federal Shipping	\$25.73
Federal Shipping	\$66.29
Federal Shipping	\$7,607.00
Federal Shipping	\$13.84
Federal Shipping	\$125.77
Federal Shipping	\$84.81
Federal Shipping	\$229.24
Federal Shipping	\$12.76
Federal Shipping	\$22.77
Federal Shipping	\$21.18
Federal Shipping	\$257.62
Federal Shipping	\$7.56
Federal Shipping	\$1.61
Federal Shipping	\$24.69
Federal Shipping	\$150.15
Federal Shipping	\$64.50

The screenshot shows the phpPgAdmin interface with the following details:

- Window title: phpPgAdmin - SQL - Mozilla Fir...
- Buttons: SQL, Find
- Database: Northwind
- SQL Query:

```
SELECT v."Firma", b."Frachtkosten"  
FROM "Bestellungen" b, "Versandfirmen" v  
WHERE b."Versandueber"=v."FirmenNr"  
ORDER BY v."Firma";
```
- Buttons: Go, Explain, Reset



- Lösung: „GROUP BY“ + Aggregationsfunktion
- „GROUP BY“ teilt die Tupel der Ergebnisrelation in Gruppen ein, so dass sich eine Äquivalenzrelation bzgl. der angegebenen Attributmenge ergibt
- „GROUP BY“ ist optional und steht zwischen „WHERE“ und „ORDER BY“
- Aggregationsfunktionen
 - `count (*)` – Zählen der Tupel der Ergebnisrelation
 - `min (NameNichtGruppierterAttributes)` – gib jeweils das Minimum der Elemente in der angegebenen Spalte der Tupel jeder Gruppe zurück
 - `max (NameNichtGruppierterAttributes)` – genau so
 - `avg (NameNichtGruppierterAttributes)` – genau so
 - `sum (NameNichtGruppierterAttributes)` – genau so
 - `count ([distinct] NameNichtGruppierterAttributes)` – Zählen der Tupel pro Gruppe. Mit optionalem `distinct` werden zuerst die Duplikate bezüglich des angegebenen Attributes eliminiert.



Northwind: SQL: Query Results

Firma	sum
Federal Shipping	\$159,610.48
Speedy Express	\$147,780.09
United Package	\$352,484.70

3 row(s)

[Back](#) | [Create report](#)

The screenshot shows a window titled "phpPgAdmin - SQL - Mozilla Fir...". It features a "SQL" tab and a "Find" search field. The "SQL" section displays the following query:

```
Database: Northwind
SELECT v."Firma", sum(b."Frachtkosten")
FROM "Bestellungen" b, "versandfirmen" v
WHERE b."Versandueber"=v."FirmenNr"
GROUP BY v."Firma"
ORDER BY v."Firma";
```

Below the query are three buttons: "Go", "Explain", and "Reset".



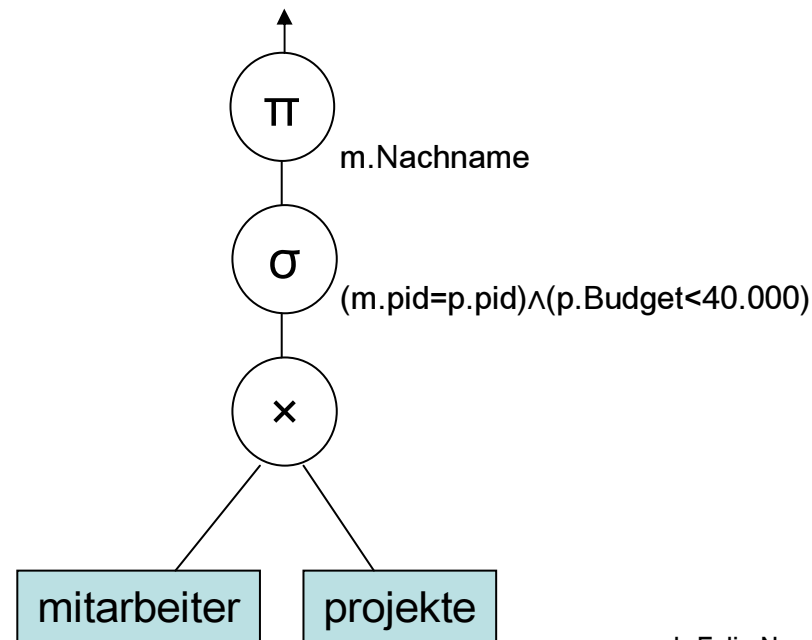
▪ SQL-Anfrage:

```
SELECT m.Nachname
FROM mitarbeiter m, projekt p
WHERE m.p_id = p.p_id
AND p.Budget < 40.000
```

▪ Ausdruck in relationaler Algebra:

$$\pi_{m.Nachname}(\sigma_{((m.pid=p.pid)\wedge(p.Budget<40.000))}(m \times p))$$

▪ Anfragebaum:

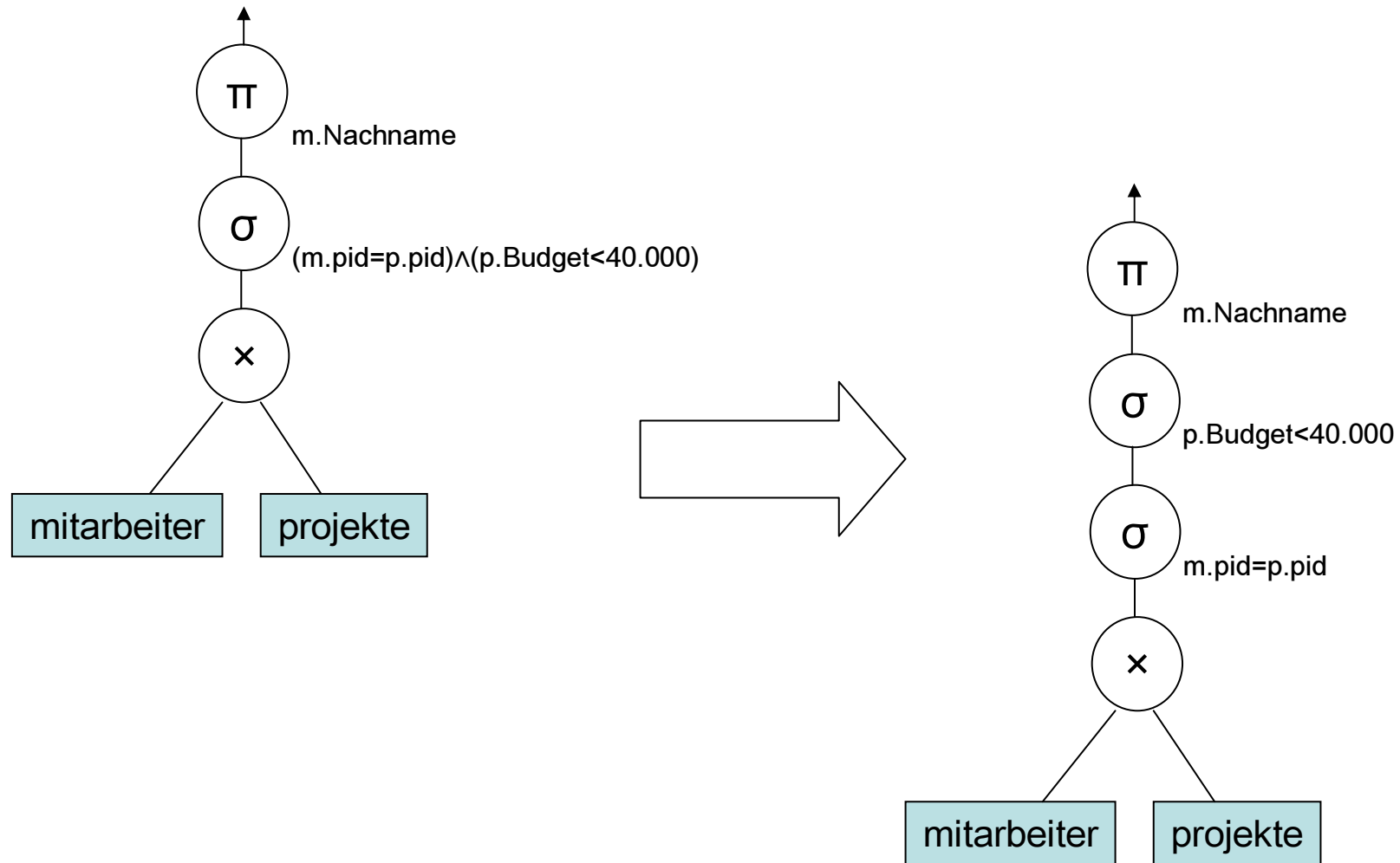


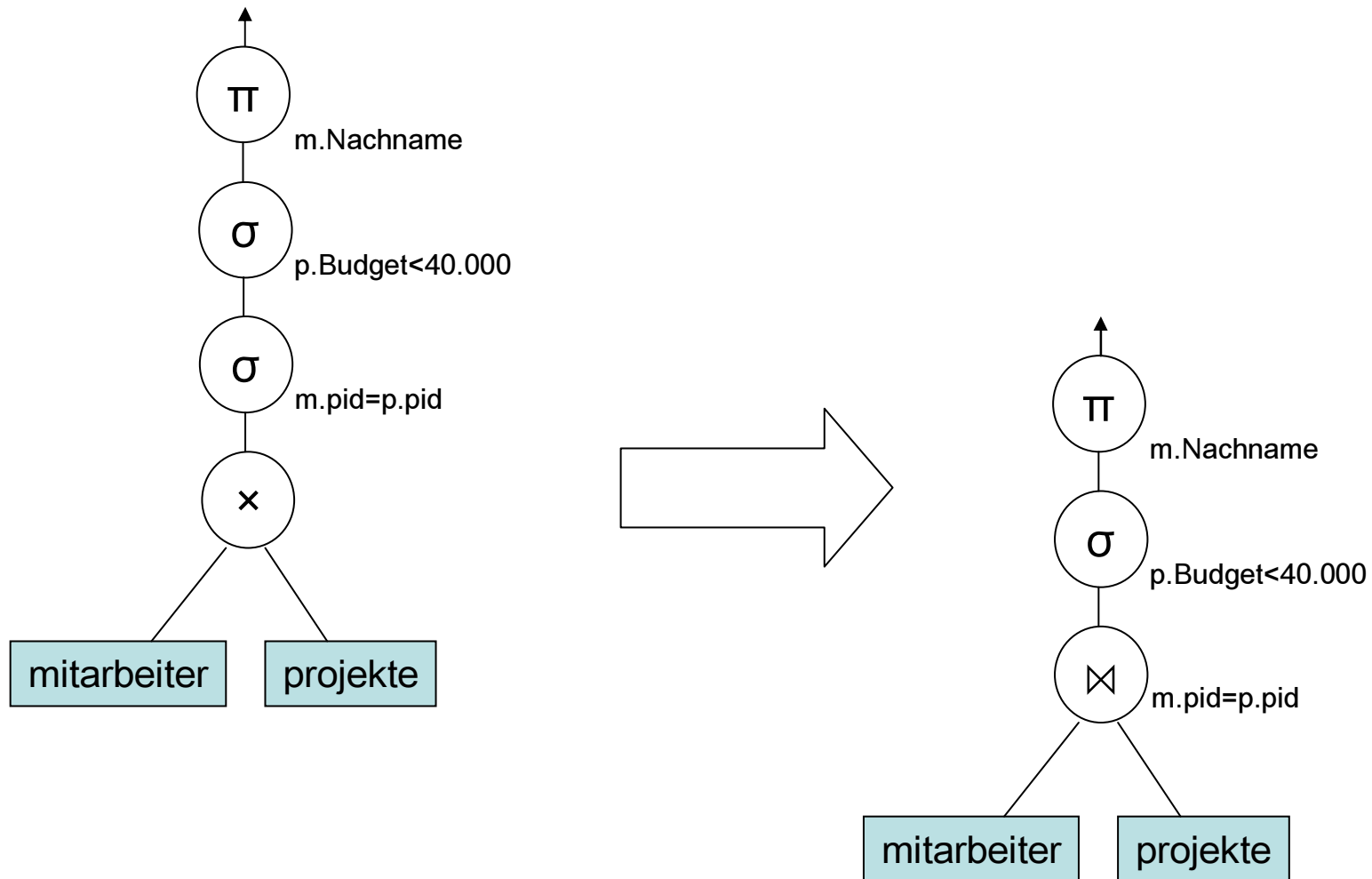
nach Felix Naumann: Crashkurs Informationssysteme, siehe http://www.informatik.hu-berlin.de/mac/lehre/WS03/CK_RDBMS.pdf

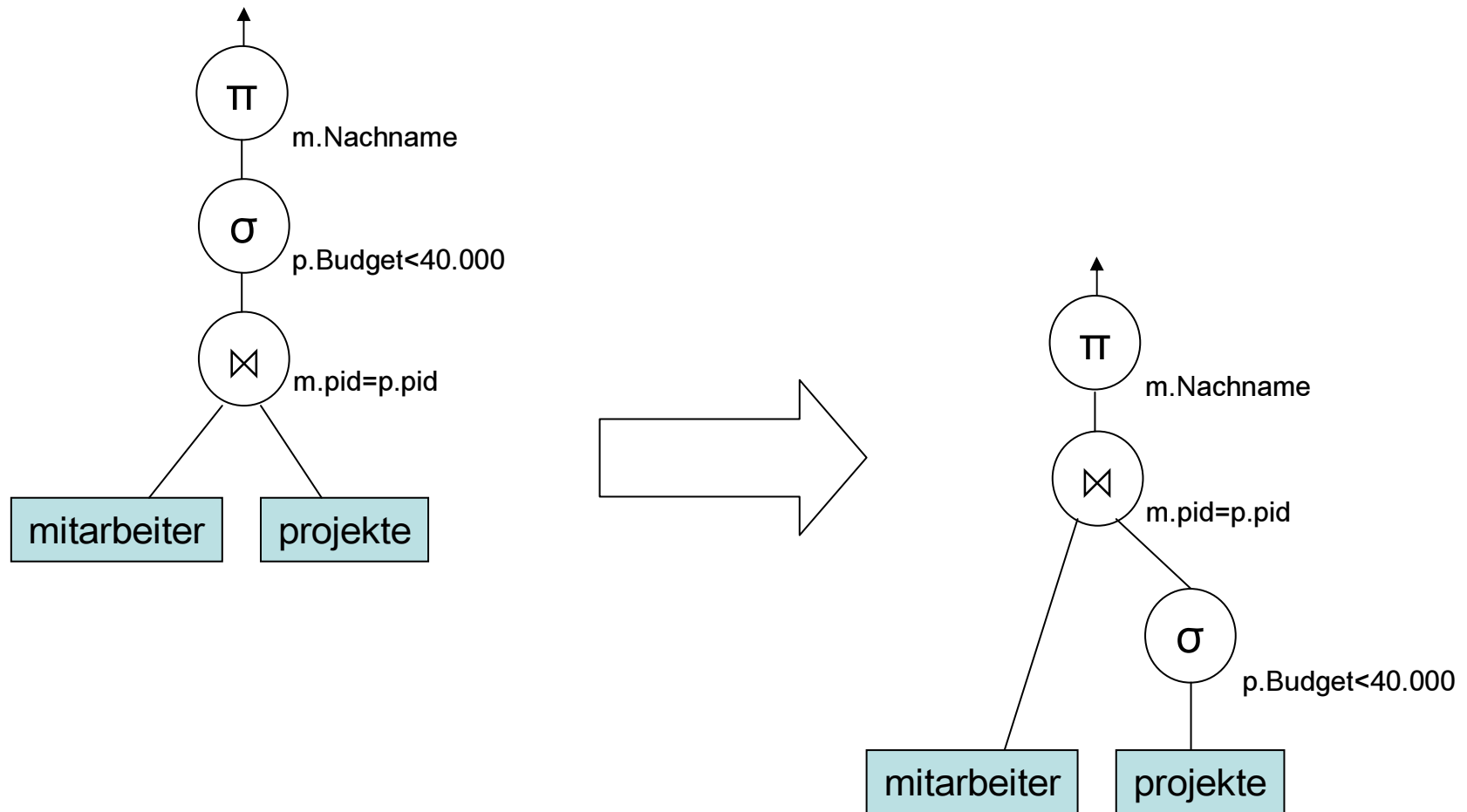


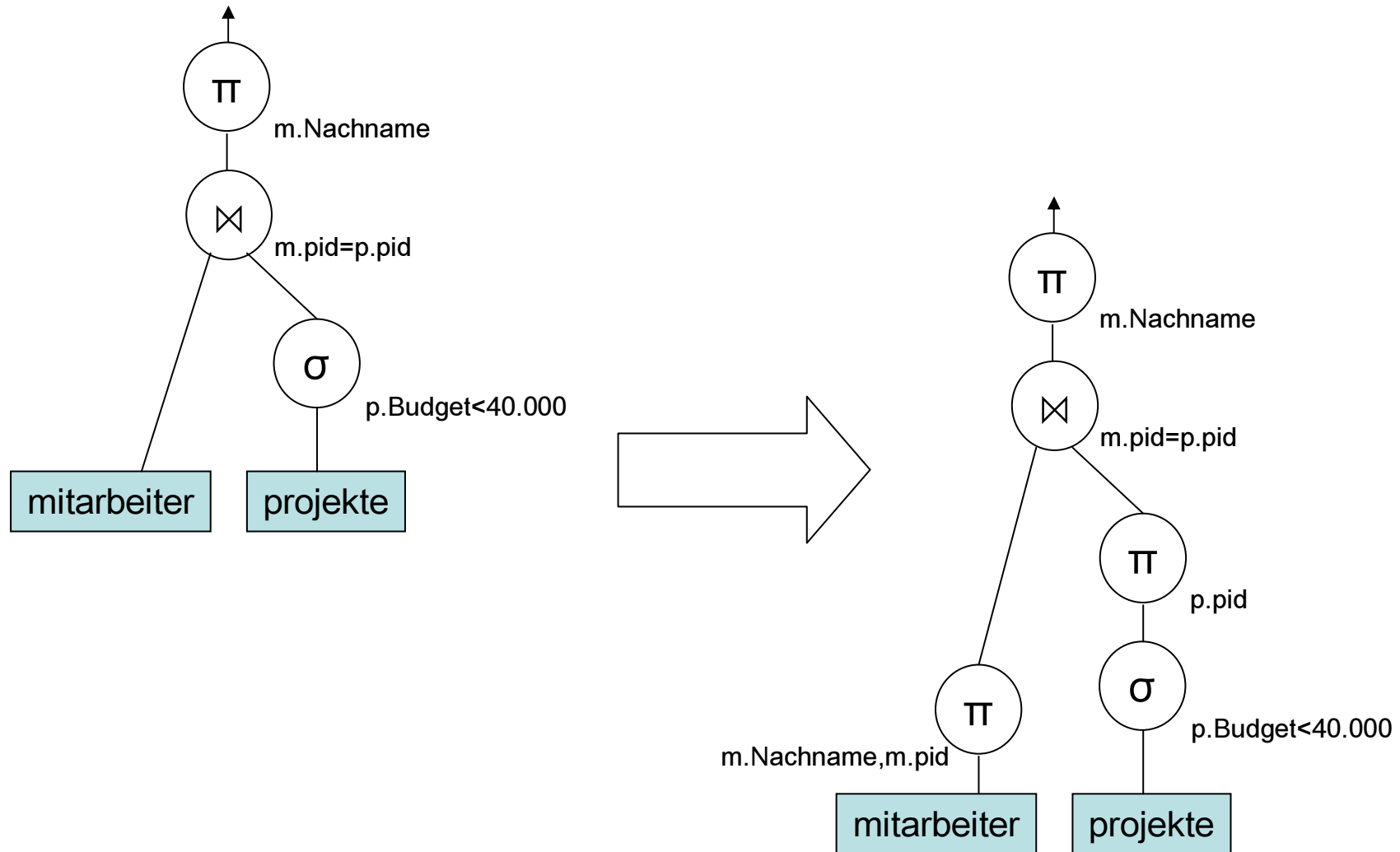
- Transformation der internen Darstellung
 - zur effizienteren Berechnung
 - ohne Semantik zu verändern: suche nach äquivalenten Ausdrücken, die
 - kleine Zwischenergebnisse erfordern.
- Beispiele für anwendbare Transformationen
 - Kommutativität von σ und \times $\sigma_P(\sigma_Q(\rho)) = \sigma_{P \wedge Q}(\rho) = \sigma_Q(\sigma_P(\rho))$
 - Assoziativität von σ und \times $(\rho_1 \times \rho_2) \times \rho_3 = \rho_1 \times (\rho_2 \times \rho_3)$
 - Projektionskaskaden: $\pi_{L_1}(\pi_{L_2}(\rho)) = \pi_{L_1}(\rho)$, wenn $L_1 \subseteq L_2$
 - Zusammenfassung von σ und \times zu Join-Operation (entspr. Def.)
 - π über σ ziehen: $\pi_{m.Nachname}(\sigma_{m.pid=p.pid}(m \times p)) = \pi_{m.Nachname}(\sigma_{m.pid=p.pid}(\pi_{m.Nachname, m.pid, p.pid}(m \times p)))$
 - π über \bowtie ziehen: $\pi_{m.Nachname}(m \bowtie_{m.pid=p.pid} p) = \pi_{m.Nachname}(\pi_{m.Nachname, m.pid}(m) \bowtie \pi_{p.pid}(p))$
 - σ über \bowtie ziehen: $\sigma_{p.Budget < 40.000}(p \bowtie m) = (\sigma_{p.Budget < 40.000}(p)) \bowtie m$
 - Beweise: Übung!

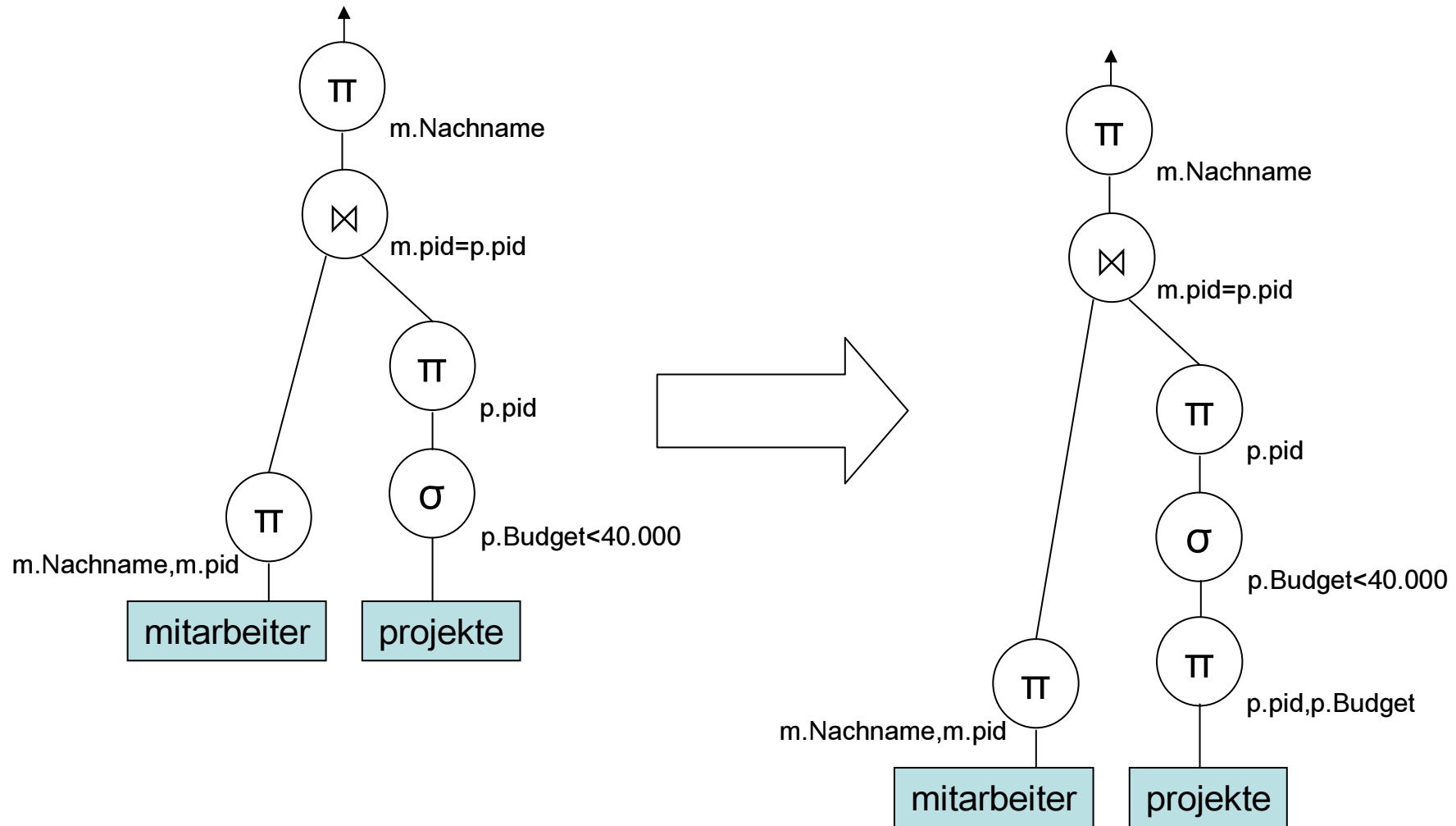












- SQL arbeitet auf der theoretischen Grundlage relationaler Algebra
- SQL arbeitet jedoch mit Vielfachmengen, genauer gesagt mit Listen:
 - ⇒ Duplikate können in (Quell- und Ergebnis-) Relationen enthalten sein (Hinweis: `DISTINCT`)
 - ⇒ Die Tupel einer Relation haben eine Reihenfolge, die man beeinflussen kann (`ORDER BY`)
- SQL ist eine deklarative Sprache, man sagt dem DBMS also was man haben will, nicht, wie das DBMS das Ergebnis bauen soll
 - ⇒ Daraus ergeben sich, wie eben gesehen, Optimierungsmöglichkeiten für das DBMS
- Zentrales Konzept der SQL ist der JOIN
 - ⇒ Lernen, verstehen, üben, üben, üben!

