

# EINFÜHRUNG IN DATENBANKEN

## ÜBUNGSERGÄNZUNG SQL

PROF. DR. RER. NAT. ALEXANDER VÖß

FH AACHEN  
UNIVERSITY OF APPLIED SCIENCES

10.10.2021

# Übersicht

---

- Unit 0x00: Beispieldaten
- Unit 0x01: Select
- Unit 0x02: Join
- Unit 0x03: Group Funktions
- Unit 0x04: Subselects
- Unit 0x05: Schemas und Tabellen
- Unit 0x06: Zeit- und Datumsfunktionen
- Unit 0x07: Views
- Unit 0x08: Transaktionen
- Unit 0x09: Daten anlegen, verändern, löschen
- Unit 0x0a: Trigger, Stored Procedures

# UNIT 0x00

## ALLGEMEINE INFORMATIONEN

## Anmerkungen vorab

---

Im Text ist bei RDBMS von Attributen/Spalten, Tabellen, Schlüsseln, Schemata und Datenbanken die Rede. Hier eine kleine Begriffsbestimmung vorab.

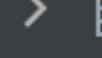
- **Attribute, Tabellen:** Nebenstehende Tabelle namens produkt modelliert Produkte aus dem Sortiment der MATSE-MHIST AG. Die Spalten der Tabelle sind die sogenannten Attribute, also Eigenschaften, eines jeden Produkts. Jede Zeile ist eine Entität/ein Objekt – hier ein Produkt.
- **Datentypen:** Attribute sind typisiert. Standardtypen in MySQL sind beispielsweise:
  - INT : klassischer Integer
  - FLOAT (M, D) : Floating-Point, (M, D) bedeutet M Ziffern total, D Nachkommastellen
  - VARCHAR (M) : Text mit maximaler Länge von M Zeichen
  - DATETIME : Zeitpunkt in der Form 'YYYY-MM-DD hh:mm:ss'

	bezeichnung	einheit	stueckpreis
1	Spinat	PK	1.99
2	Vier Käse Pizza	ST	2.39
3	Spinatpizza	ST	2.29
4	Fischstäbchen	PK	1.99

## Anmerkungen vorab

---

- **Schlüssel, ID:** Fast immer besitzt eine Entität eine Art eindeutige Kennung oder ID, auch Schlüssel genannt. Diese identifiziert die konkrete Entität und ist eindeutig.
- **Fremdschlüssel:** Ein Attribut in einer Tabelle, das einen Schlüssel in einer anderen Tabelle darstellt, nennt sich Fremdschlüssel. Die genaue Definition und der Sinn dahinter folgen noch.
- **Schema/Datenbank:** Tabellen sind in einem Schema zusammengefasst. Je nach DBMS heisst es auch einfach Datenbank statt Schema.
- **Indexe:** Um einen schnellen Zugriff auf Daten zu ermöglichen, werden, automatisch oder intendiert, zusätzliche Strukturen angelegt, z.B. B-trees.

 matse_mhist
>  abteilung
>  arbeitet_in_als
>  besteht_aus
>  bestellung
>  kennt
>  kunde
>  mitarbeiter
>  produkt
>  ressource

## Anmerkungen vorab

---

- **GROSS/klein:** Ob Befehle, Attribute und Tabellennamen groß - oder kleingeschrieben werden können oder müssen, hängt vom DBMS, den dortigen Einstellungen und dem darunter liegenden Betriebssystem ab. Für die Prüfung ist es nicht relevant.
- **NULL:** Neben dem typischen Wertebereich eines jeden Datentyps gibt es einen weiteren Wert, den ein Attribut besitzen kann: `NULL`. Die Bedeutung des Werts `NULL` gibt nur der Kontext, häufig wird der Wert einfach für 'keine Angabe' oder 'noch nicht spezifiziert' verwendet.  
Die Definition der Spalte einer Tabelle kann `NULL` zulassen oder verhindern, aber wenn der Wert erlaubt ist, erfordert es oft eine Sonderbehandlung in den Abfragen.
- **Skript:** Die SQL-Befehle einer Unit sind als `*.sql`-File im Ilias verfügbar, so dass man die Kommandos nicht abzutippen braucht sondern direkt ausprobieren kann.

## Beispieldaten MATSE-MHIST

---

### Idee

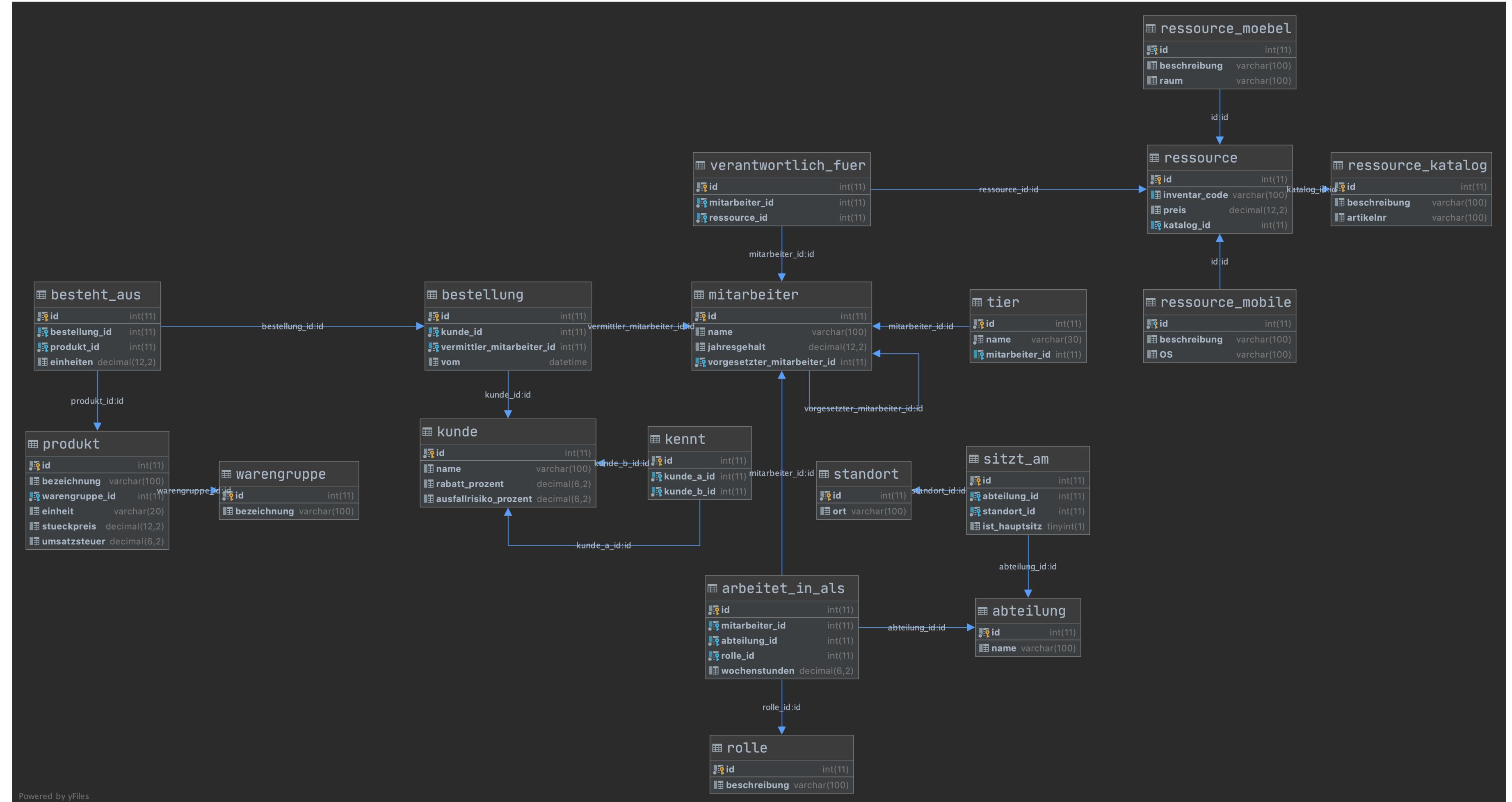
- Die MATSE-MHIST AG ist ein fiktives Unternehmen mit einzelnen Abteilungen und Mitarbeitern an unterschiedlichen Standorten, die Dinge des täglichen Bedarfs liefert, wenn mal keine Zeit ist, einzukaufen. Typische Beispiele sind TK-Pizzen oder die neuste c't.
- Die Datenbank enthält eine Menge von Daten und Relationen und dient zur Übung der SQL-Kommados.

**Bitte importieren Sie sie (Anleitung im Video zur Installation).**

## Beispieldaten MATSE-MHIST

### Modellierte Themenfelder

- Abteilungen/ Standorte
- Bestellungen/ Produkte
- Mitarbeiter/ Vorgesetzte
- Betriebsmittel/ Ressourcen
- und noch einige mehr.

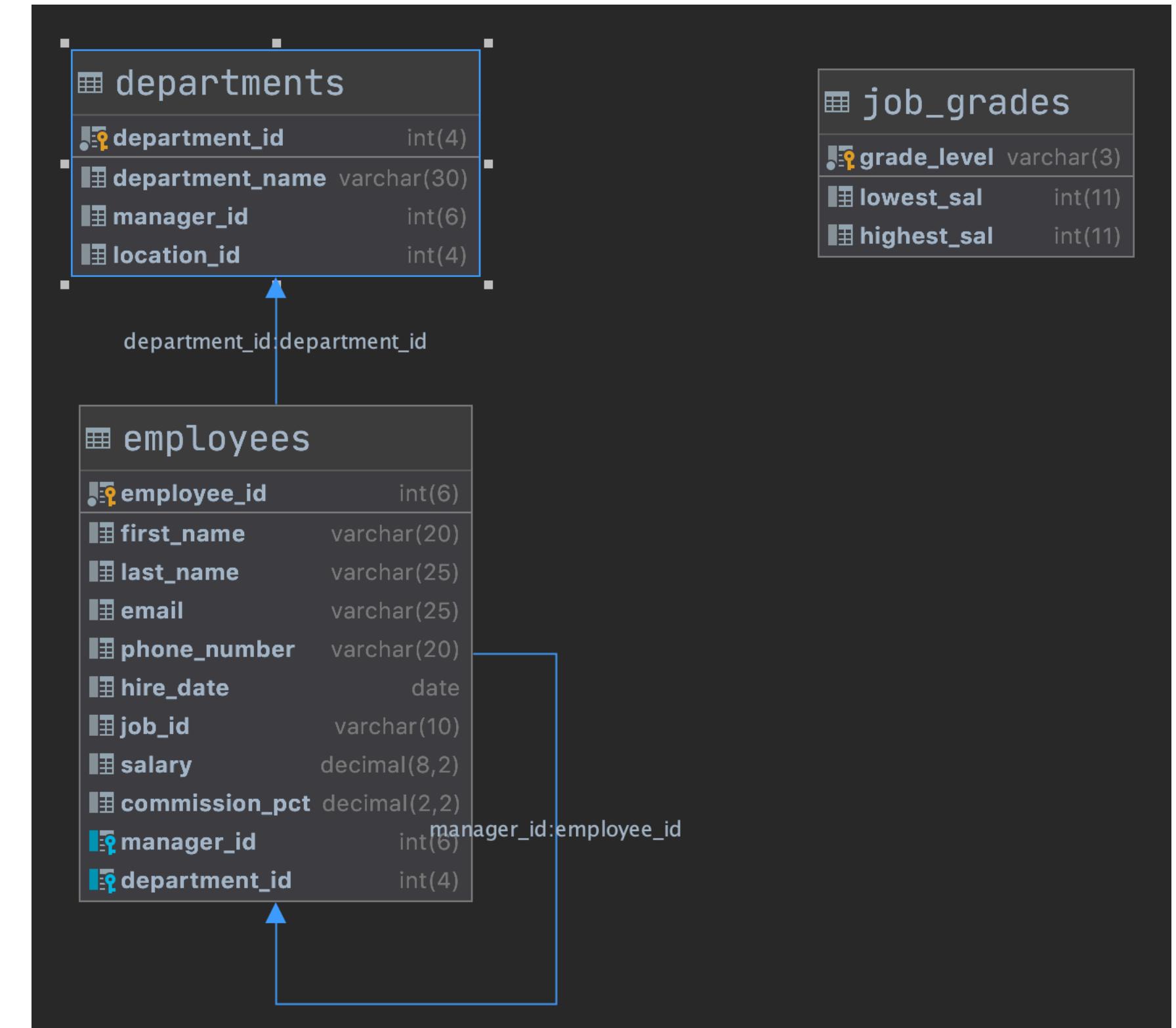


Powered by yFiles

## Beispieldaten ORACLE-HR

### Idee

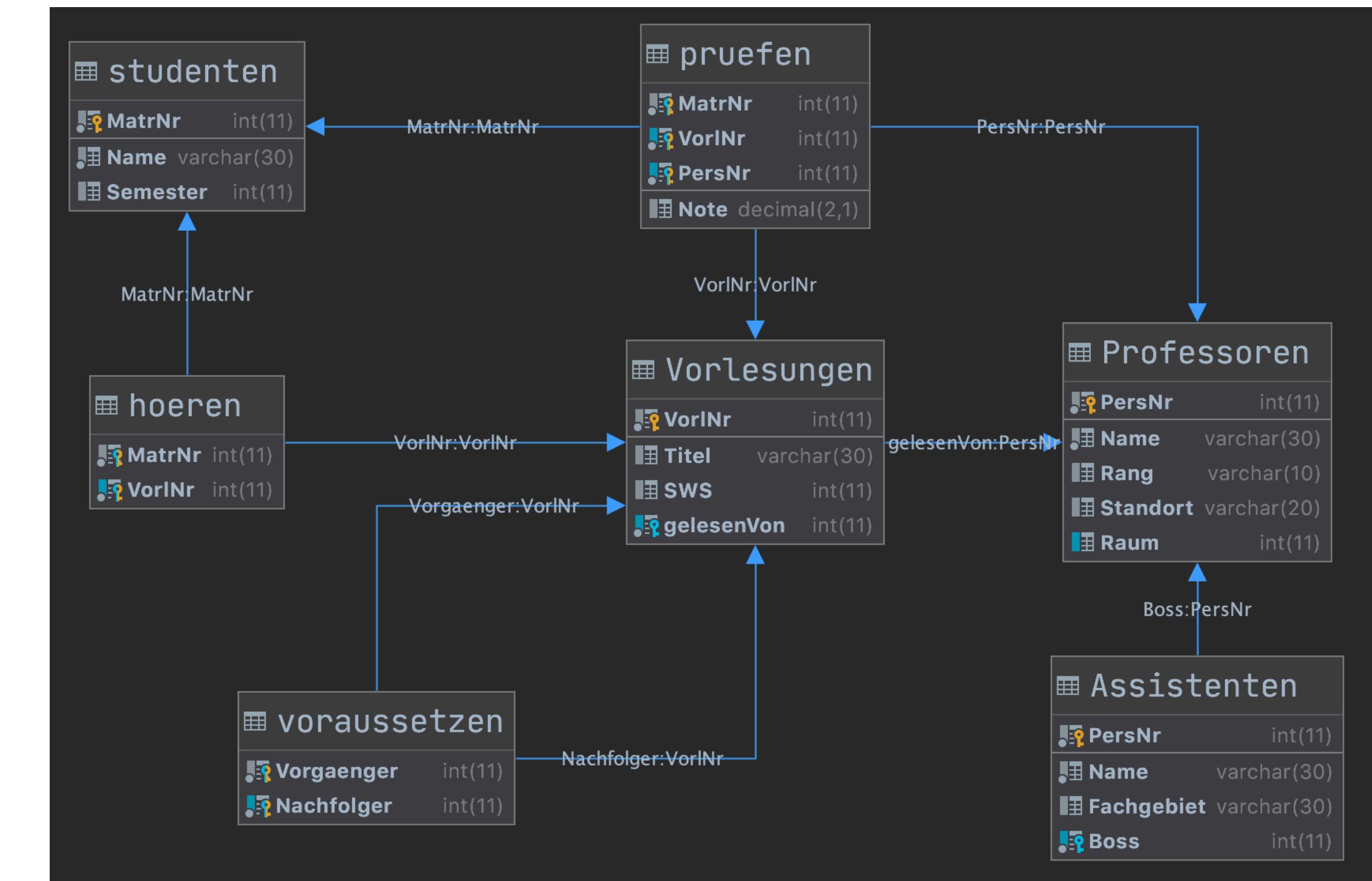
- Die ORACLE-HR genannte Datenbank ist inhaltlich den Übungsmaterialien von Oracle entnommen und kann als Grundlage dienen, die zugehörigen Aufgaben in anderen DBMS zu bearbeiten.
- Für unsere Übungen benötigen wir diese Daten nicht.



## Beispieldaten KEMPER-DB

### Idee

- Die Kemper-Datenbank modelliert Aspekte eines Uni-Betriebs mit Professoren, Assistenten, Prüfungen usw.
- Diese Datenbank wird häufig auch in den Prüfungen für die SQL-Kommandos verwendet und kommt zuweilen in der Vorlesung vor. Die Tabellen sollten Sie daher grundsätzlich kennen.



# UNIT 0x01

## SELEKTION, PROJEKTION

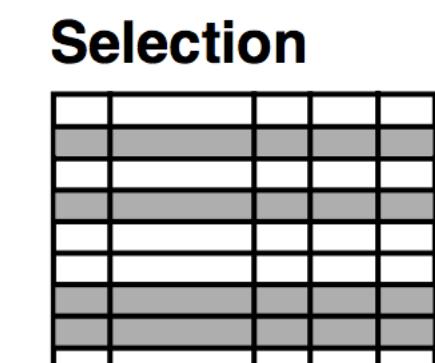
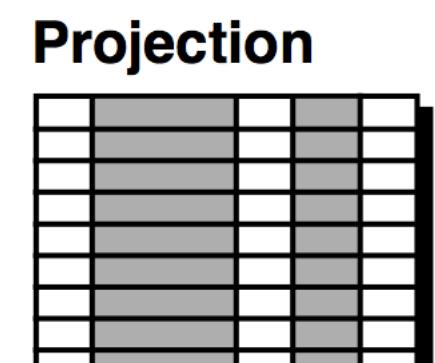
## Select

### Lernziel

- Datenanfragen mittels select formulieren.

### Projektion und Selektion

- "Projektion" wählt Spalten durch Angabe von Attributen der Tabelle.
- "Selektion" filtert Zeilen durch Angabe eines Kriteriums, welches je Zeile erfüllt sein muss.



Oracle Dokumentation

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr] ...
  [into_option]
  [FROM table_references
    [PARTITION partition_list]]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
  [HAVING where_condition]
  [WINDOW window_name AS (window_spec)
    [, window_name AS (window_spec)] ...]
  [ORDER BY {col_name | expr | position}
    [ASC | DESC], ... [WITH ROLLUP]]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [into_option]
  [FOR {UPDATE | SHARE}
    [OF tbl_name [, tbl_name] ...]
    [NOWAIT | SKIP LOCKED]
    | LOCK IN SHARE MODE]
  [into_option]
  into_option: {
    INTO OUTFILE 'file_name'
      [CHARACTER SET charset_name]
      export_options
    | INTO DUMPFILE 'file_name'
    | INTO var_name [, var_name] ...
  }
```

MySQL Dokumentation

## Select – Use

---

### Intention

- Default-Schema (je nach DBMS auch Default-Datenbank) mit USE festlegen.

### Anmerkungen

- Ist ein Default-Schema mit USE festgelegt, so kann bei Tabellen in SQL-Ausdrücken die explizite Angabe des Schemas entfallen.
- Die Erklärung des Befehls SELECT folgt sofort (kurz: Alle Daten abfragen), aber mit vorherigem USE kann hier die Tabelle produkt auch ohne die Angabe von matse\_mhist angesprochen werden.

```
# default schema
USE matse_mhist;

# all data
SELECT * FROM matse_mhist.produkt;
SELECT * FROM produkt;
```

## Select - \*/From

---

### Intention

- Alle Datensätze (keine Selektion) und alle Attribute (keine Projektion) einer Tabelle abfragen.

### Anmerkungen

- '\*' selektiert alle Attribute einer Tabelle.
- Die Spaltenüberschriften sind die Attributnamen bzw. Spalten der Tabelle.
- Die Reihenfolge der Datensätze ist hier nicht vorgegeben.

The screenshot shows a database interface with a query editor and a results table. The query in the editor is:

```
SELECT * FROM produkt;
```

The results table has the following data:

		id	bezeichnung	warengruppe_id	einheit	stueckpreis	umsatzsteuer
1	1	Spinat		1	PK	1.99	0.07
2	2	Vier Käse Piz...		1	ST	2.39	0.07
3	3	Spinatpizza		1	ST	2.29	0.07
4	4	Fischstäbchen		1	PK	1.99	0.07
5	5	Nudelpfanne		1	PK	7.29	0.07

Beispiel matse\_mhist

## Select – Tabellenalias

---

### Intention

- Tabellen in SELECT mit Alias versehen.

### Anmerkungen

- Tabellen können mit einem Aliasnamen versehen werden, hier `P`, der stattdessen verwendet wird.
- Das macht komplexere Ausdrücke nicht nur kürzer, sondern löst vor allem mögliche Mehrdeutigkeiten auf.
- Attribute bekommen dann ein '`P.`'-Prefix, siehe '`P.*`'.

The screenshot shows a database interface with a query editor and a results table. The query in the editor is:

```
SELECT P.*| FROM produkt P;
```

The results table is titled "matse\_mhist.produkt" and contains the following data:

	id	bezeichnung	waren...	einheit	stueck...	umsatzs...
1	1	Spinat		1 PK	1.99	0.07
2	2	Vier Käse Piz...		1 ST	2.39	0.07
3	3	Spinatpizza		1 ST	2.29	0.07
4	4	Fischstäbchen		1 PK	1.99	0.07

Beispiel matse\_mhist

## Select – Projektion/Spaltenalias

### Intention

- Nur *bestimmte* Attribute abfragen (Projektion).

### Anmerkungen

- Spaltenbezeichnungen, hier z.B. ID, kann man explizit angeben.
- AS ist optional.

The screenshot shows a MySQL command-line interface. The query window contains the following SQL code:

```
SELECT
    id AS 'ID',
    bezeichnung 'Produkt',
    stueckpreis 'Preis'
FROM produkt;
```

The results window shows the output of the query:

ID	Produkt	Preis
1	Spinat	1.99
2	Vier Käse Pizza	2.39
3	Spinatpizza	2.29
4	Fischstäbchen	1.99
5	Nudelsoße	2.29

Beispiel matse\_mhist

## Select – Selektion/Where

---

### Intention

- Nur Attribute oder Ausdrücke einer Tabelle abfragen, die eine Bedingung (hinter WHERE) erfüllen (Selektion).

### Anmerkungen

- Die Bedingung, hier `stueckpreis=1.99`, testet auf Gleichheit, allerdings gelten grundsätzlich natürlich weiterhin die Vorbehalte beim Vergleich von Fließkommazahlen mit Blick auf Rundungsfehler.

The screenshot shows a database interface with a query editor and a results table. The query in the editor is:

```
SELECT bezeichnung, stueckpreis FROM produkt  
WHERE stueckpreis=1.99;
```

The results table, titled "matse\_mhist.produkt", displays the following data:

	bezeichnung	stueckpreis
1	Spinat	1.99
2	Fischstäbchen	1.99
3	Fleischwurst	1.99
4	Chips	1.99
5	Nudeln in Soße	1.99

Beispiel matse\_mhist

## Select – Funktionen

### Intention

- Arithmetische Ausdrücke und Funktionen (z.B. round, concat) verwenden (Stichwort 'SQL Function and Operator Reference').

### Anmerkungen

- Syntax und mögliche Variationen am besten in der Dokumentation nachsehen.

The screenshot shows a database query editor with a dark theme. At the top, there is a code editor window containing the following SQL query:

```
SELECT
    id 'ID',
    concat(bezeichnung, '[' ,einheit, ']') 'Produkt',
    stueckpreis 'Preis',
    round(stueckpreis/(1.0+umsatzsteuer),2) 'exkl. USt.'
FROM produkt;
```

Below the code editor is a table titled "Produkt" with three columns: "ID", "Produkt", and "Preis". The table contains the following data:

ID	Produkt	Preis	exkl. USt.
1	1 Spinat[PK]	1.99	1.86
2	2 Vier Käse Pizza[ST]	2.39	2.23
3	3 Spinatpizza[ST]	2.29	2.14
4	4 Fischstäbchen[PK]	1.99	1.86
5	5 Nudelnpfanne[PK]	3.29	3.07

Beispiel matse\_mhist

## Select - Operatoren/NULL

### Intention

- Arithmetische Ausdrücke wie 'is NULL' und Funktionen mit NULL verwenden.

### Anmerkungen

- Achtung: Ausdrücke sind NULL, wenn einer der Operanden NULL ist.
- coalesce gibt den ersten nicht-NULL Parameter zurück.

The screenshot shows a database interface with a code editor and a results table. The code in the editor is:

```
SELECT
    name,
    ausfallrisiko_prozent,
    100-ausfallrisiko_prozent 'Erfuellung1',
    if(ausfallrisiko_prozent is NULL,
        100.00,100-ausfallrisiko_prozent) 'Erfuellung2',
    coalesce(100-ausfallrisiko_prozent,100.00) 'Erfuellung3'
FROM kunde;
```

The results table has columns: name, ausfallrisiko\_prozent, Erfuellung1, Erfuellung2, and Erfuellung3. The data is as follows:

	name	ausfallrisiko_prozent	Erfuellung1	Erfuellung2	Erfuellung3
4	Daimler	<null>	<null>	100.00	100.00
5	Deutsche Bank	<null>	<null>	100.00	100.00
6	Sparkasse	<null>	<null>	100.00	100.00
7	Börse	<null>	<null>	100.00	100.00
8	E.ON	<null>	<null>	100.00	100.00
9	Infinion	<null>	<null>	100.00	100.00
10	Lufthansa	<null>	<null>	100.00	100.00
11	RWE	2.00	98.00	98.00	98.00
12	SAP	<null>	<null>	100.00	100.00

Beispiel matse\_mhist

## Select - Operatoren/NULL

### Intention

- Operatoren wie 'is NULL' bzw. 'is not NULL' und Klammerung in WHERE-Bedingungen verwenden.

### Anmerkungen

- Achtung bei NULL-Werten. Das Ergebnis einer arithmetischen Operation mit NULL ergibt ebenfalls NULL.

```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE (stueckpreis<1.00 or stueckpreis>5.00)
    and warengruppe_id=11;
SELECT name, ausfallrisiko_prozent FROM kunde
    WHERE ausfallrisiko_prozent is not null;
```

Output matse\_mhist.produkt

bezeichnung	stueckpreis
Bild	0.90
GameStar	6.50

```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE (stueckpreis<1.00 or stueckpreis>5.00)
    and warengruppe_id=11;
SELECT name, ausfallrisiko_prozent FROM kunde
    WHERE ausfallrisiko_prozent is not null;
```

Output matse\_mhist.kunde

name	ausfallrisiko_prozent
RWE	2.00

Beispiel matse\_mhist

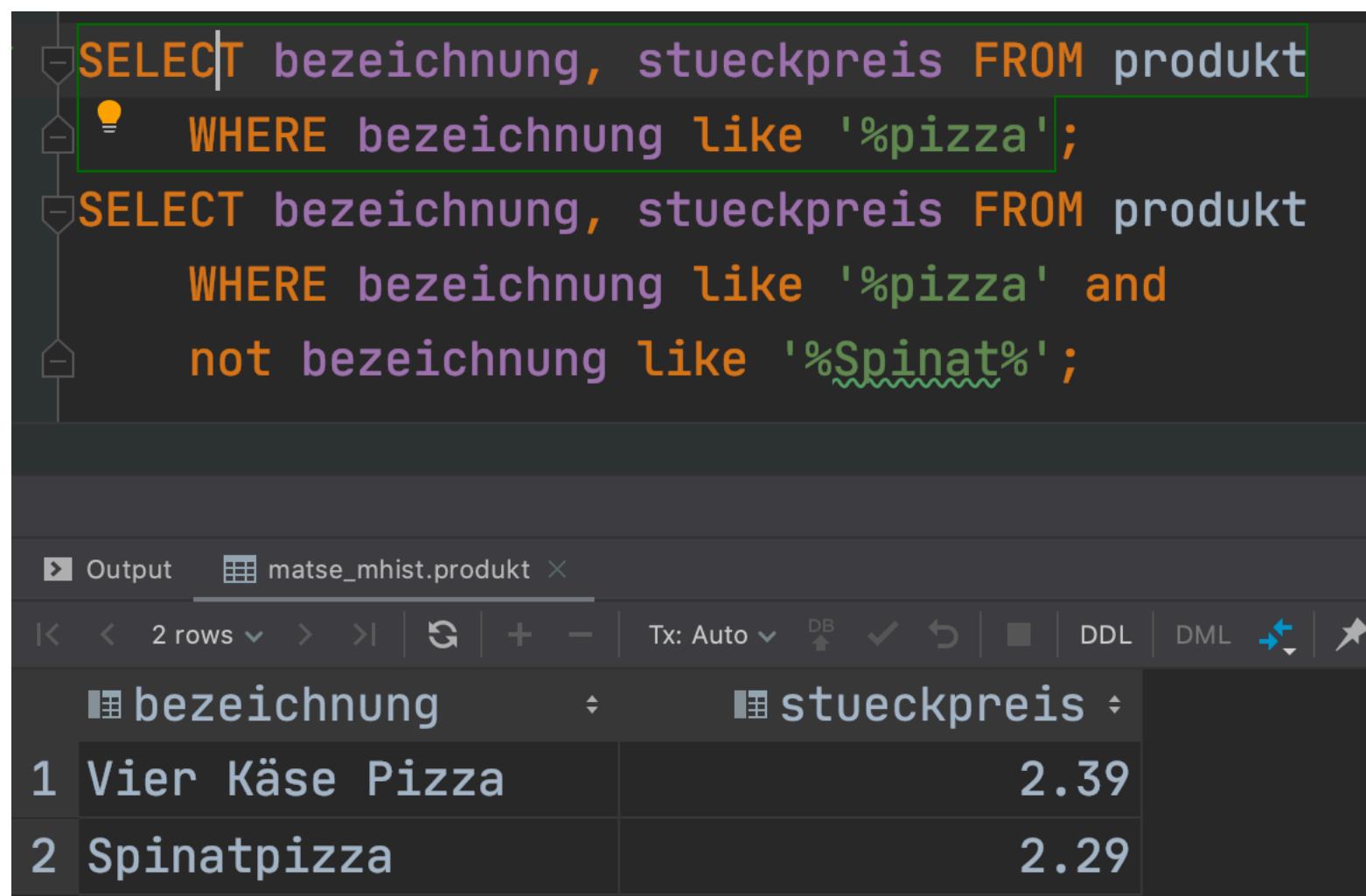
## Select - Operatoren

### Intention

- Operatoren wie 'like', 'not', 'and' in WHERE-Bedingungen verwenden.

### Anmerkungen

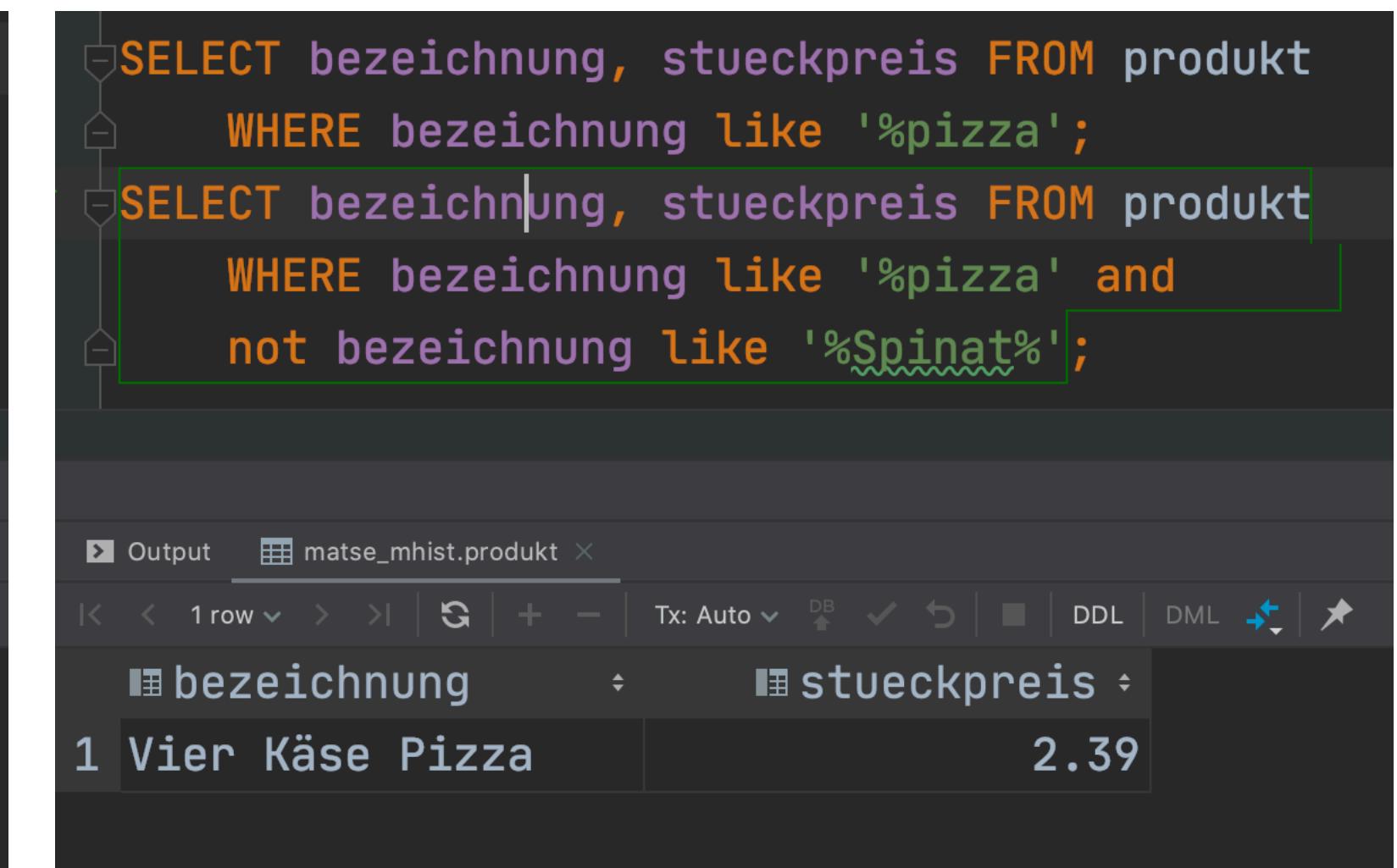
- like dient dem String-Vergleich, allerdings berücksichtigt oder ignoriert die Operation je nach DBMS Groß- und Kleinschreibung.
- Die Wildcard '%' lässt alle Zeichen zu (auch 0),
- '\_' genau ein beliebiges Zeichen.



```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE bezeichnung like '%pizza';
SELECT bezeichnung, stueckpreis FROM produkt
WHERE bezeichnung like '%pizza' and
not bezeichnung like '%Spinat%';
```

Output matse\_mhist.produkt

bezeichnung	stueckpreis
1 Vier Käse Pizza	2.39
2 Spinatpizza	2.29



```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE bezeichnung like '%pizza';
SELECT bezeichnung, stueckpreis FROM produkt
WHERE bezeichnung like '%pizza' and
not bezeichnung like '%Spinat%';
```

Output matse\_mhist.produkt

bezeichnung	stueckpreis
1 Vier Käse Pizza	2.39

Beispiel matse\_mhist

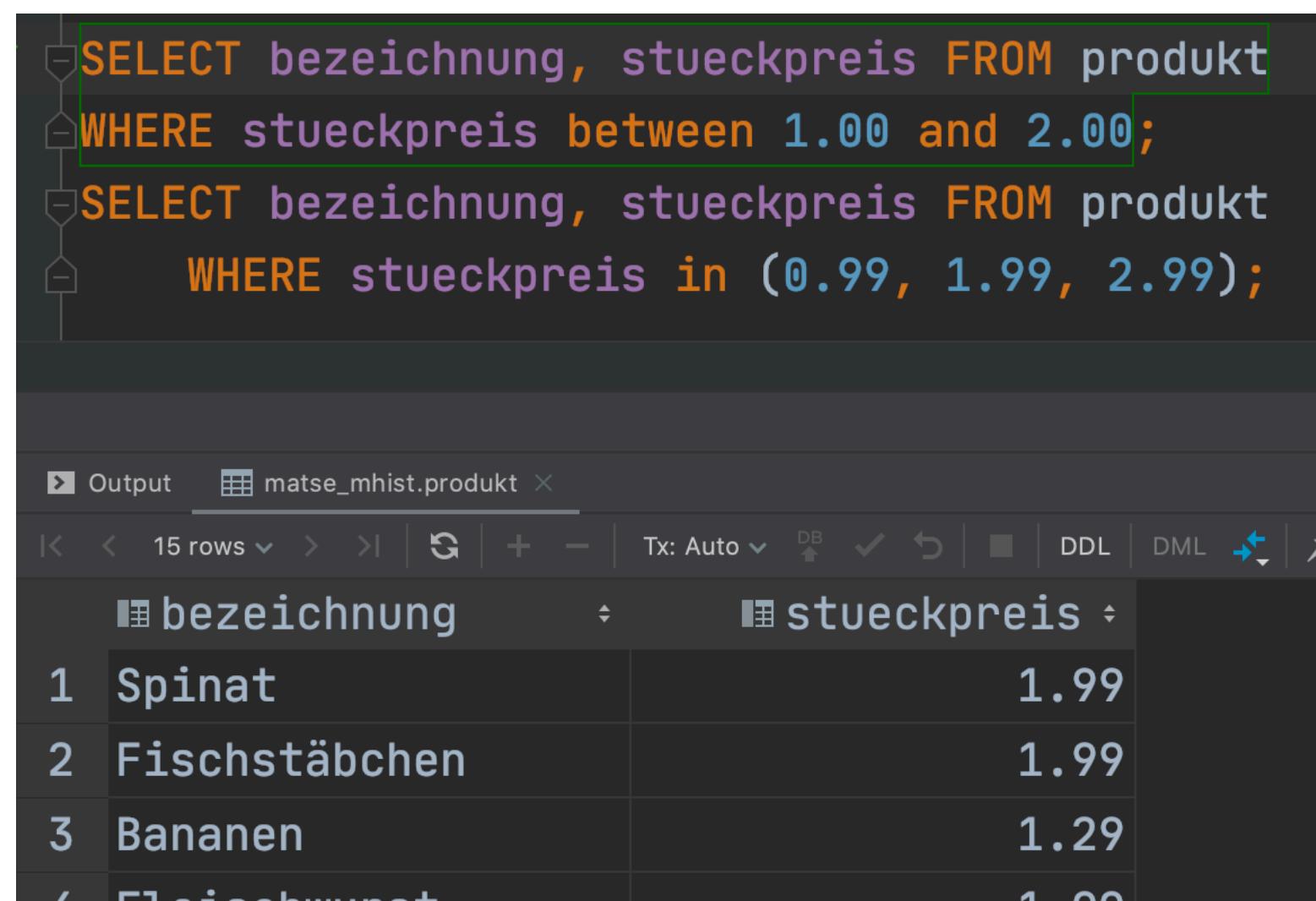
## Select - Operatoren

### Intention

- Operatoren wie 'between...and', 'in' in WHERE-Bedingungen verwenden.

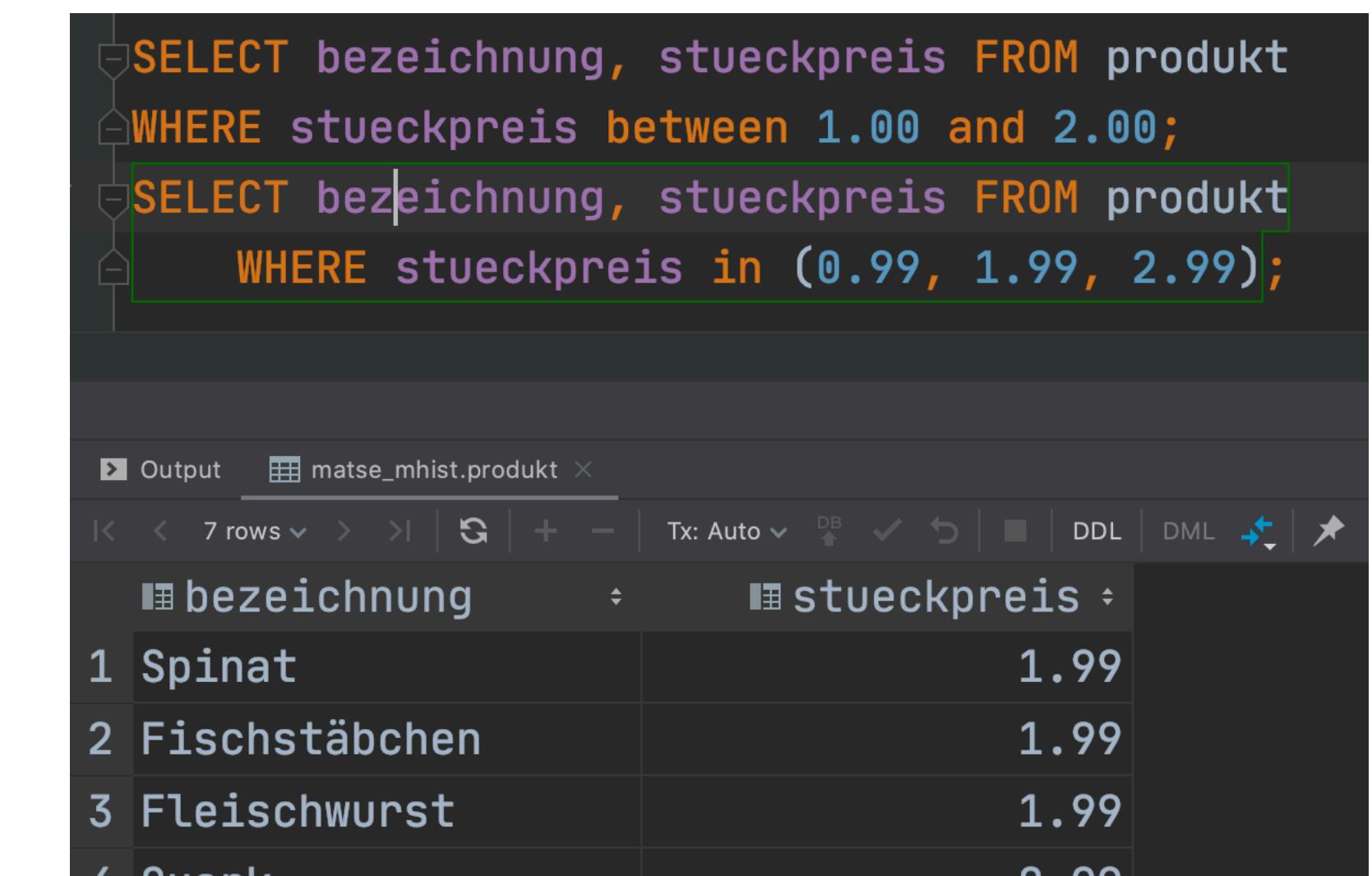
### Anmerkungen

- Grenzen bei 'between...and' sind inklusive.



```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE stueckpreis between 1.00 and 2.00;
SELECT bezeichnung, stueckpreis FROM produkt
WHERE stueckpreis in (0.99, 1.99, 2.99);
```

bezeichnung	stueckpreis
Spinat	1.99
Fischstäbchen	1.99
Bananen	1.29
Fleischwurst	1.99
Quark	0.99



```
SELECT bezeichnung, stueckpreis FROM produkt
WHERE stueckpreis between 1.00 and 2.00;
SELECT bezeichnung, stueckpreis FROM produkt
WHERE stueckpreis in (0.99, 1.99, 2.99);
```

bezeichnung	stueckpreis
Spinat	1.99
Fischstäbchen	1.99
Bananen	1.29
Fleischwurst	1.99
Quark	0.99

Beispiel matse\_mhist

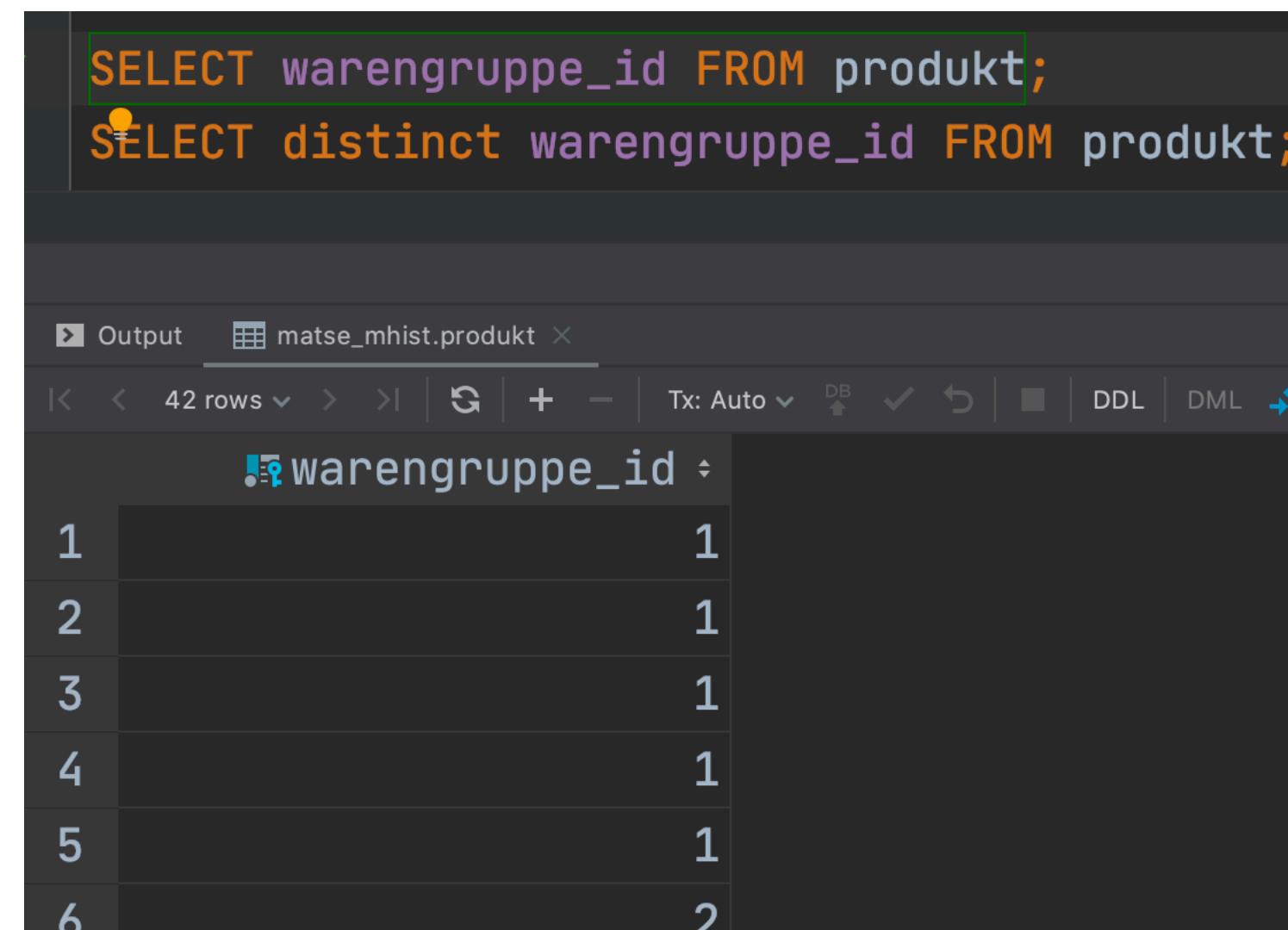
## Select – Distinct

### Intention

- Doppelte Daten sollen nicht mit berücksichtigt werden.

### Anmerkungen

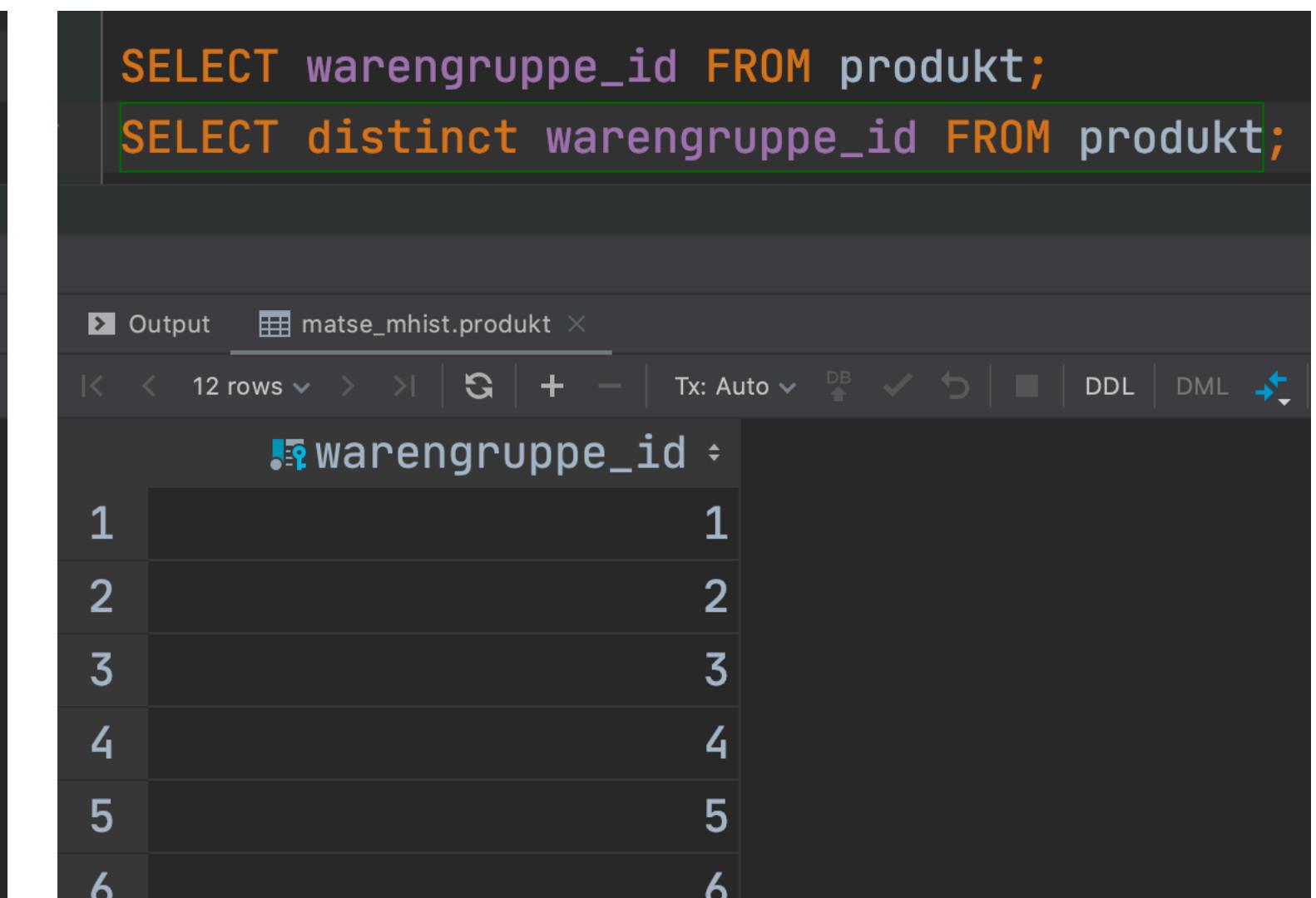
- `distinct` verwenden, sonst können Ergebniszellen mehrfach vorkommen.
- Hier bricht die Analogie zu einer mathematischen Menge.



```
SELECT warengruppe_id FROM produkt;
SELECT distinct warengruppe_id FROM produkt;
```

Output matse\_mhist.produkt

warengruppe_id
1
2
3
4
5
6
1
1
1
1
1
2



```
SELECT warengruppe_id FROM produkt;
SELECT distinct warengruppe_id FROM produkt;
```

Output matse\_mhist.produkt

warengruppe_id
1
2
3
4
5
6
1
2
3
4
5
6

Beispiel matse\_mhist

# Unit 0x01 - Selektion, Projektion

## Select - Count

### Intention

- Ermittlung der Anzahl Datensätze.

### Anmerkungen

- '\*' zählt alle Datensätze, wogegen die Angabe eines Attributes bewirkt, dass nur Datensätze ungleich NULL gezählt werden.

id	name	rabatt_prozent	ausfallrisiko_prozent
1	2 Bayer	1.00	<null>
2	26 BMW	1.00	<null>
3	27 Commerzbank	0.00	<null>
4	28 Daimler	1.00	<null>
5	29 Deutsche Bank	0.00	<null>
6	30 Sparkasse	10.00	<null>
7	31 Börse	0.00	<null>
8	32 E.ON	1.00	<null>
9	33 Infineon	2.00	<null>
10	34 Lufthansa	2.00	<null>
11	35 RWE	1.00	2.00
12	36 SAP	2.00	<null>

Beispiele matse\_mhist

```
select count(*) from kunde;  
select count(rabatt_prozent) from kunde;  
select count(ausfallrisiko_prozent) from kunde;
```

Output count(\*):int  
1 row  
1 12

```
select count(*) from kunde;  
select count(rabatt_prozent) from kunde;  
select count(ausfallrisiko_prozent) from kunde;
```

Output count(rabatt\_prozent):int  
1 row  
1 12

```
select count(*) from kunde;  
select count(rabatt_prozent) from kunde;  
select count(ausfallrisiko_prozent) from kunde;
```

Output count(ausfallrisiko\_prozent):int  
1 row  
1 1

## Select – Ordnung

### Intention

- Ergebnismenge nach Kriterien sortieren (`order by`), auf- (`asc`) oder absteigend (`desc`).
- Ergebnismenge nach mehreren Kriterien sortieren.

```

SELECT bezeichnung, stueckpreis FROM produkt
WHERE warengruppe_id in (1,3,4) ORDER BY stueckpreis desc;
SELECT bezeichnung, stueckpreis FROM produkt
WHERE warengruppe_id in (1,3,4) ORDER BY stueckpreis asc,
bezeichnung desc;
  
```

Output matse\_mhist.produkt

	bezeichnung	stueckpreis
1	Nudelpfanne	3.29
2	Vier Käse Pizza	2.39
3	Frikadellen	2.35
4	Spinatpizza	2.29
5	Spinat	1.99
6	Fleischwurst	1.99
7	Fischstäbchen	1.99
8	Ouark	0.99

	bezeichnung	stueckpreis
1	Milch	0.79
2	Joghurt	0.98
3	Quark	0.99
4	Spinat	1.99
5	Fleischwurst	1.99
6	Fischstäbchen	1.99
7	Spinatpizza	2.29
8	Frikadellen	2.35

### Anmerkungen

- In der unteren Ergebnismenge sind die 1.99-Artikel dem Kommando entsprechend absteigen alphabetisch geordnet.

Beispiel matse\_mhist

## Select – Interaktiv

---

### Intention

- Interaktive Verwendung von SELECT.

### Anmerkungen

- Eine (Pseudo)Tabelle `dual` ist in den DBMS notwendig, in denen z.B. eine Angabe einer Tabelle im SELECT-Befehl notwendig ist, etwa bei Oracle.

The screenshot shows a database interface with two SQL queries in the code editor:

```
select now(), 1+2*3, rand();
select now(), 1+2*3, rand() FROM dual;
```

The interface includes tabs for "Output" and "Result 61". The "Output" tab shows the execution of the queries. The "Result" tab displays the results in a grid:

	now()	1+2*3	rand()
1	2020-10-12 13:10:00	7	0.747948683563572

Beispiel

## Check

---

### Aufgaben

1. Geben Sie jeweils den gesamten Inhalt der Tabellen abteilung, kunde, mitarbeiter, produkt, rolle und warengruppe aus matse\_mhist aus.
2. Ermitteln Sie, wieviele verschiedene Umsatzsteuersätze in der Tabelle produkt vorhanden sind und geben Sie sie aus.
3. Welche Kunden sind dem Namen nach eine Bank? Ermitteln Sie sie und geben Sie nur den Namen sowie den jeweiligen Rabatt aus.
4. Ermitteln Sie das Monatsgehalt (12 Monate) aller Mitarbeiter, deren Vorgesetzter Mia oder Ben ist. Geben Sie der Spalte die Überschrift 'Monatsgehalt' und runden Sie das Monatsgehalt auf volle Euro. (Sehen Sie die id von Mia und Ben nach, keine Unterabfrage.)
5. Wieviele Mitarbeiternamen beginnen mit einem 'M'? Und wieviele enthalten ein 'M'? Ermitteln Sie die jeweilige Anzahl.

# UNIT 0x02

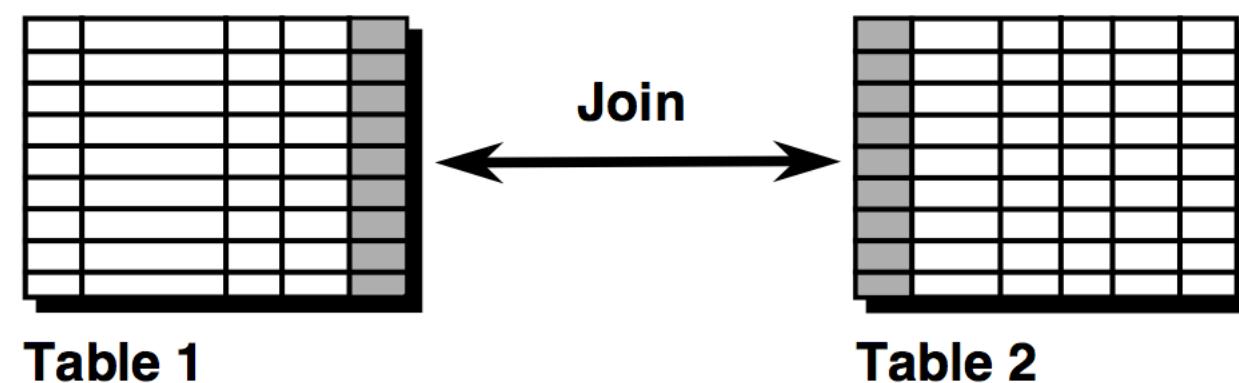
## JOINS

## Join

---

### Lernziel

- Datenabfragen über mehrere Tabellen mit sog. Joins formulieren.
- Verschiedene Arten von Joins
  - Inner,
  - Left Outer,
  - Right Outer,
  - Full Outer,
  - Cross,
  - Self und
  - Natural Joins anwenden.
- Joins werden mittels des SELECT-Befehls realisiert.



Oracle Dokumentation

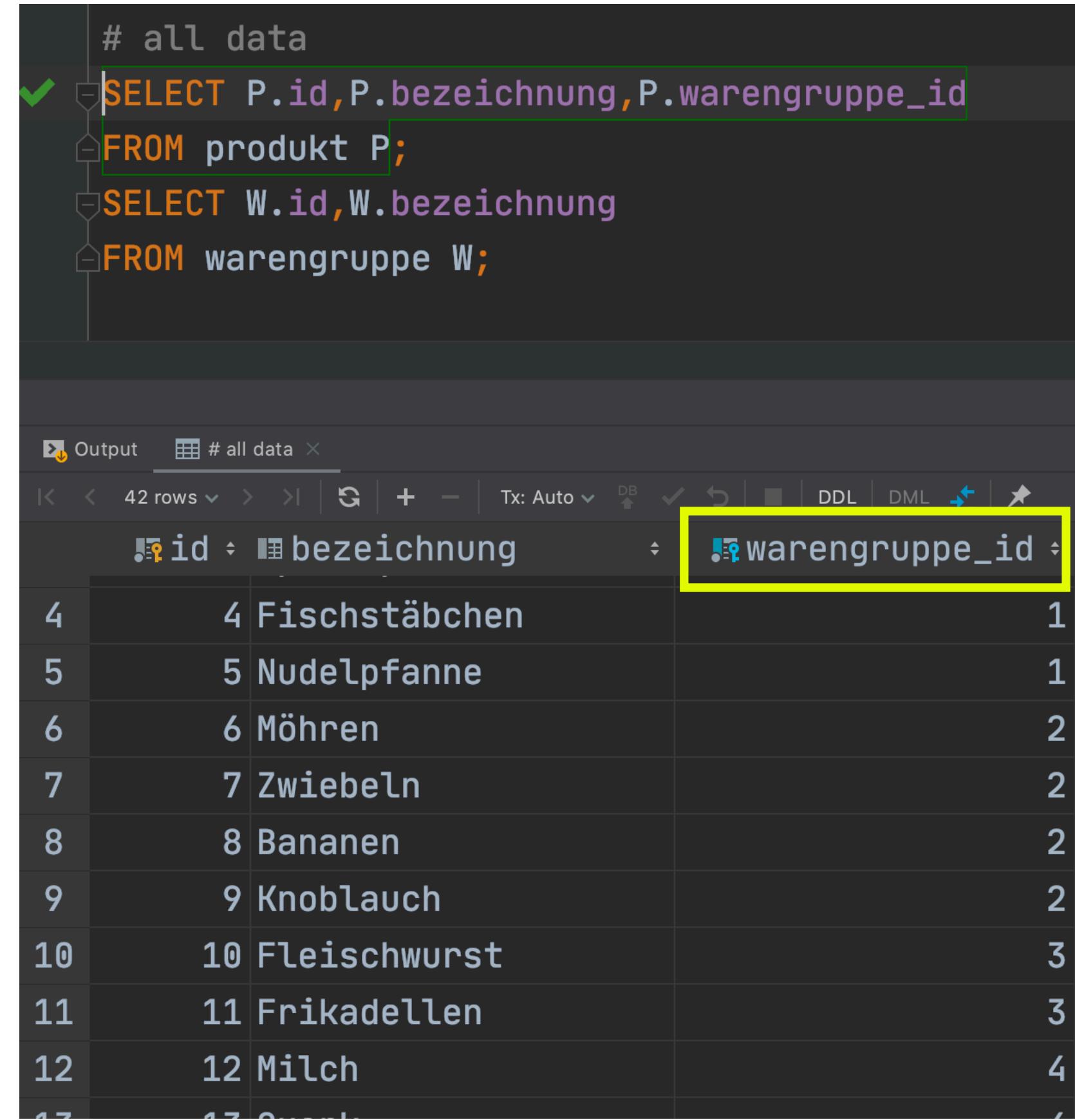
# Motivation

---

## Warengruppe zu Produkt

- Jedes Produkt ist einer Warengruppe zugeordnet (N:1) und diese Relation ist über den Fremdschlüssel `warengruppe_id` realisiert.
- Es wäre doch schön, wenn man statt dieser `id` beim Produkt auch die Warengruppe lesen könnte:

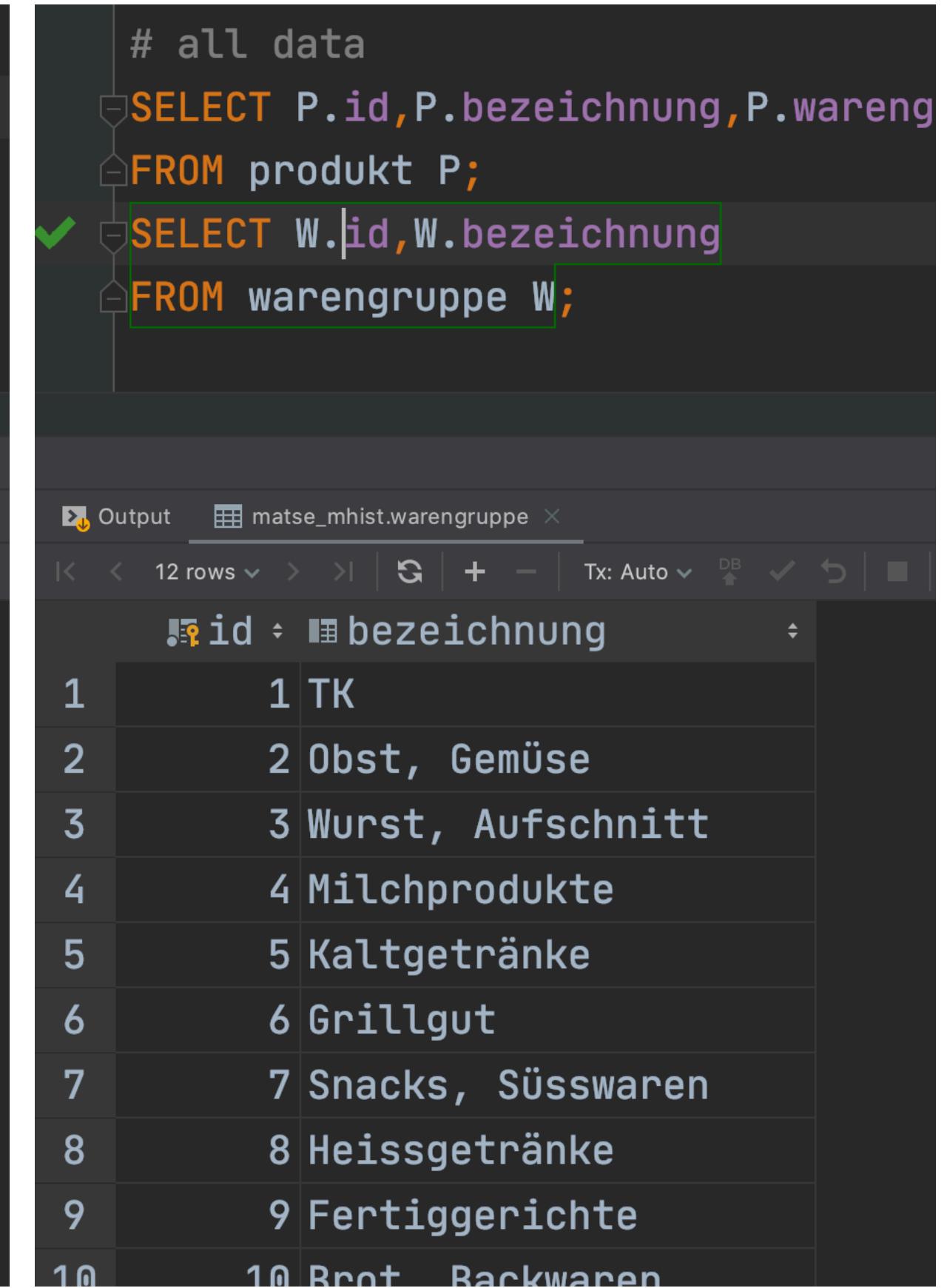
<code>id</code>	<code>P.bezeichnung</code>	<code>W.bezeichnung</code>
8	Bananen	Obst, Gemüse
9	Knoblauch	Obst, Gemüse
10	Fleischwurst	Wurst, Aufschnitt
11	Frikadellen	Wurst, Aufschnitt



```
# all data
SELECT P.id,P.bezeichnung,P.warenguppe_id
FROM produkt P;
SELECT W.id,W.bezeichnung
FROM warengruppe W;
```

Output # all data

	<code>id</code>	<code>bezeichnung</code>	<code>warenguppe_id</code>
4	4	Fischstäbchen	1
5	5	Nudelpfanne	1
6	6	Möhren	2
7	7	Zwiebeln	2
8	8	Bananen	2
9	9	Knoblauch	2
10	10	Fleischwurst	3
11	11	Frikadellen	3
12	12	Milch	4



```
# all data
SELECT P.id,P.bezeichnung,P.warenguppe_id
FROM produkt P;
SELECT W.id,W.bezeichnung
FROM warengruppe W;
```

Output matse\_mhist.warengruppe

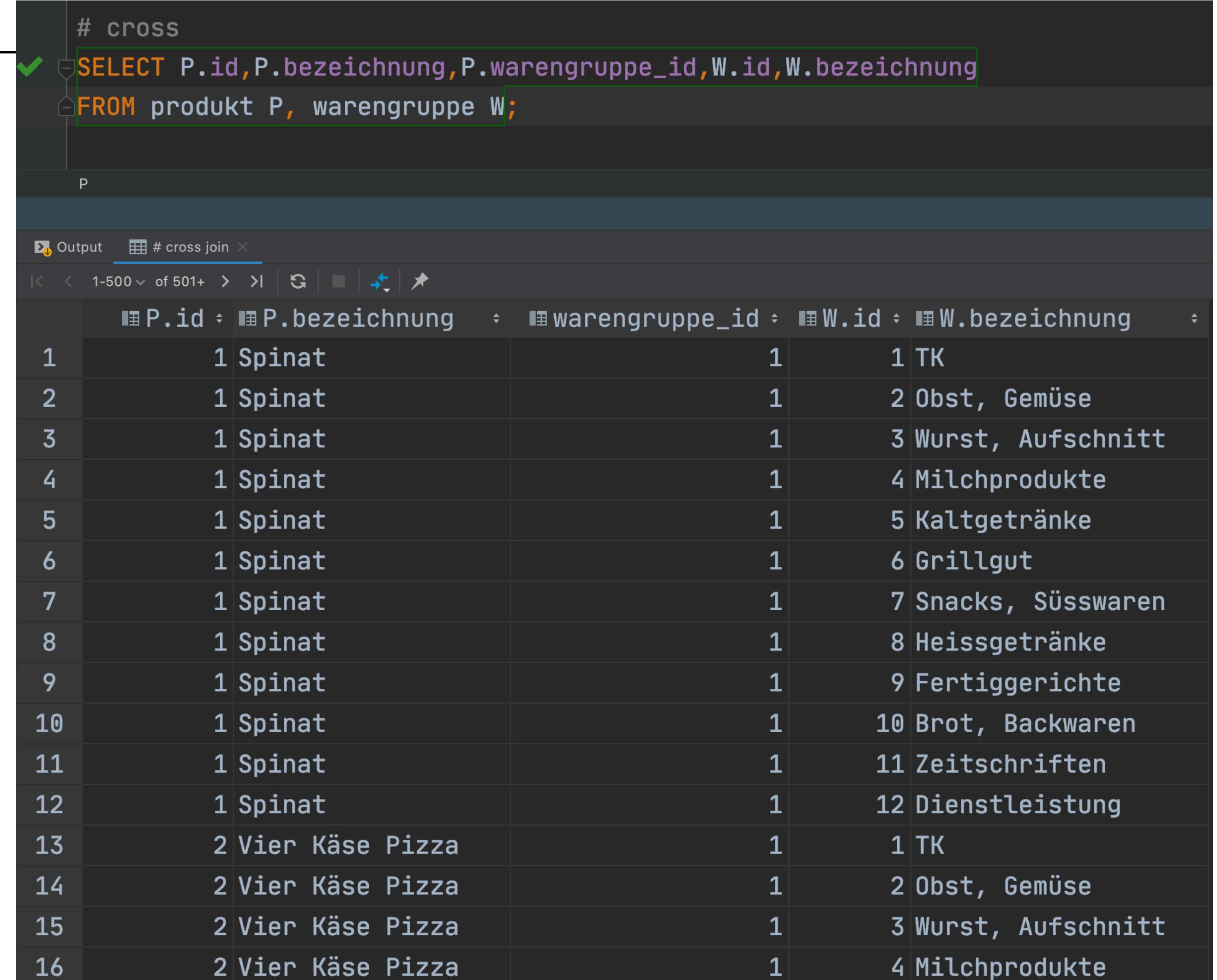
	<code>id</code>	<code>bezeichnung</code>
1	1	TK
2	2	Obst, Gemüse
3	3	Wurst, Aufschnitt
4	4	Milchprodukte
5	5	Kaltgetränke
6	6	Grillgut
7	7	Snacks, Süßwaren
8	8	Heissgetränke
9	9	Fertiggerichte
10	10	Brot, Backwaren

Beispiel matse\_mhist

## Motivation

### Warengruppe zu Produkt

- Ansatz: Gibt man beide Tabellen hinter `from` im `select`, an, so erhält man das *kartesische Produkt* – kurz: jedes Produkt mit jeder Warenguppe (ein sog. impliziter *Cross Join*).
- Wie selektiert man nun geschickt, dass nur 'passende' Zeilen in der Menge enthalten sind?  
Anders gefragt: Wie lautet die konkrete `where`-Bedingung?



The screenshot shows a database interface with a query editor and a results table.

**Query Editor:**

```
# cross
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P, warengruppe W;
```

**Results Table:**

	P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
1	1 Spinat		1	1	TK
2	1 Spinat		1	2	Obst, Gemüse
3	1 Spinat		1	3	Wurst, Aufschnitt
4	1 Spinat		1	4	Milchprodukte
5	1 Spinat		1	5	Kaltgetränke
6	1 Spinat		1	6	Grillgut
7	1 Spinat		1	7	Snacks, Süßwaren
8	1 Spinat		1	8	Heissgetränke
9	1 Spinat		1	9	Fertiggerichte
10	1 Spinat		1	10	Brot, Backwaren
11	1 Spinat		1	11	Zeitschriften
12	1 Spinat		1	12	Dienstleistung
13	2 Vier Käse Pizza		1	1	TK
14	2 Vier Käse Pizza		1	2	Obst, Gemüse
15	2 Vier Käse Pizza		1	3	Wurst, Aufschnitt
16	2 Vier Käse Pizza		1	4	Milchprodukte

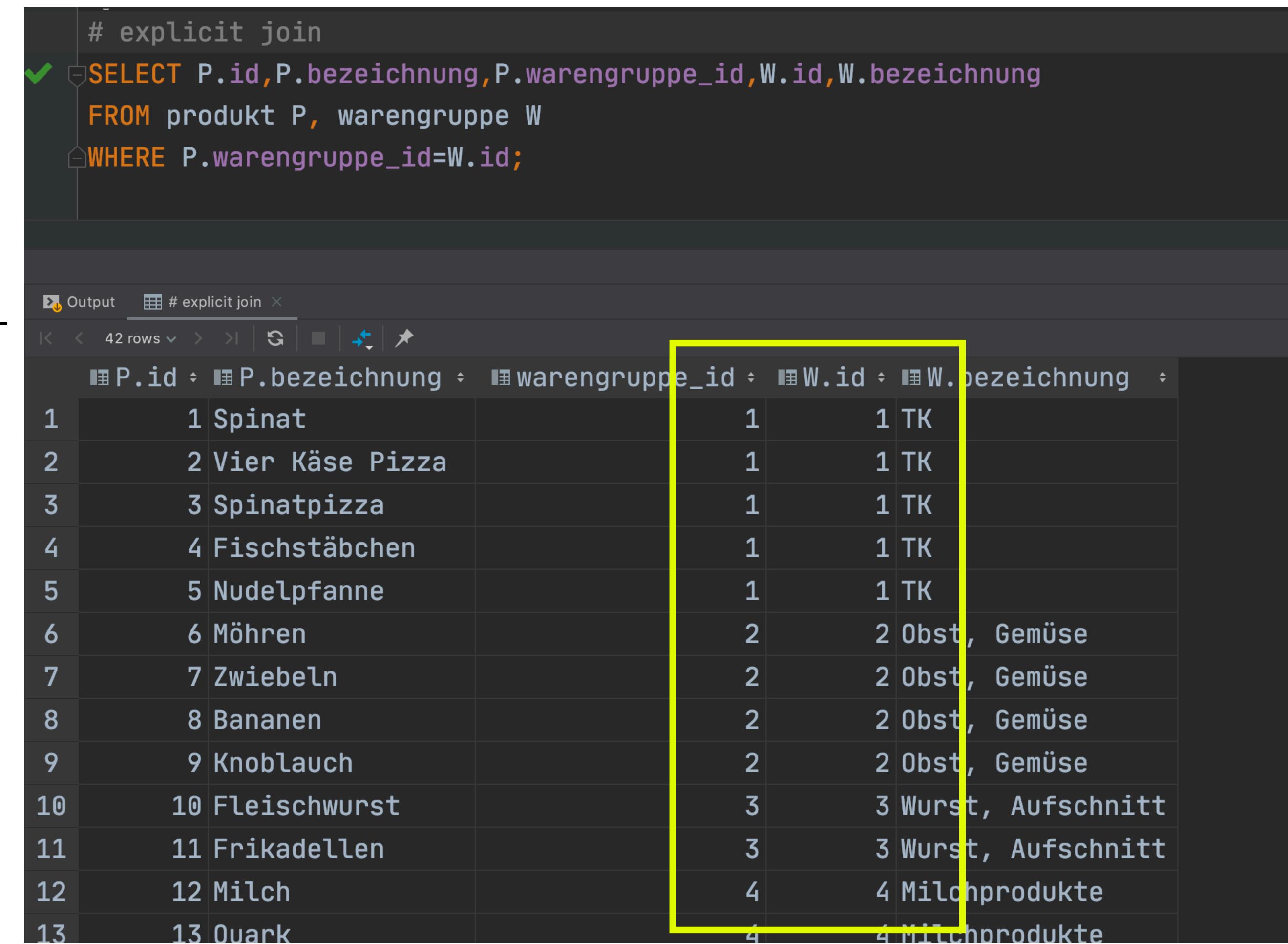
Beispiel matse\_mhist

## Motivation

---

### Warengruppe zu Produkt

- Wir interessieren uns nur für die Zeilen, wo der Fremdschlüssel `warengruppe_id` mit dem Produktschlüssel `id` übereinstimmt – das ergibt offensichtlich die korrekte Warengruppe und ist durch die `where`-Bedingung explizit ausgedrückt (nennt sich *inner join*).
- Das DBMS erkennt diesen *join* und ermittelt *nicht* erst alle Kombinationen, nur um dann fast alle wieder heraus zu filtern...



The screenshot shows a database query interface with the following code:

```
# explicit join
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P, warengruppe W
WHERE P.warengruppe_id=W.id;
```

The output window displays the results of the query:

P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
1	1 Spinat	1	1	TK
2	2 Vier Käse Pizza	1	1	TK
3	3 Spinatpizza	1	1	TK
4	4 Fischstäbchen	1	1	TK
5	5 Nudelpfanne	1	1	TK
6	6 Möhren	2	2	Obst, Gemüse
7	7 Zwiebeln	2	2	Obst, Gemüse
8	8 Bananen	2	2	Obst, Gemüse
9	9 Knoblauch	2	2	Obst, Gemüse
10	10 Fleischwurst	3	3	Wurst, Aufschnitt
11	11 Frikadellen	3	3	Wurst, Aufschnitt
12	12 Milch	4	4	Milchprodukte
13	13 Quark	4	4	Milchprodukte

A yellow box highlights the last two rows of the result set, which correspond to the products 'Milch' and 'Quark' from the 'Milchprodukte' category.

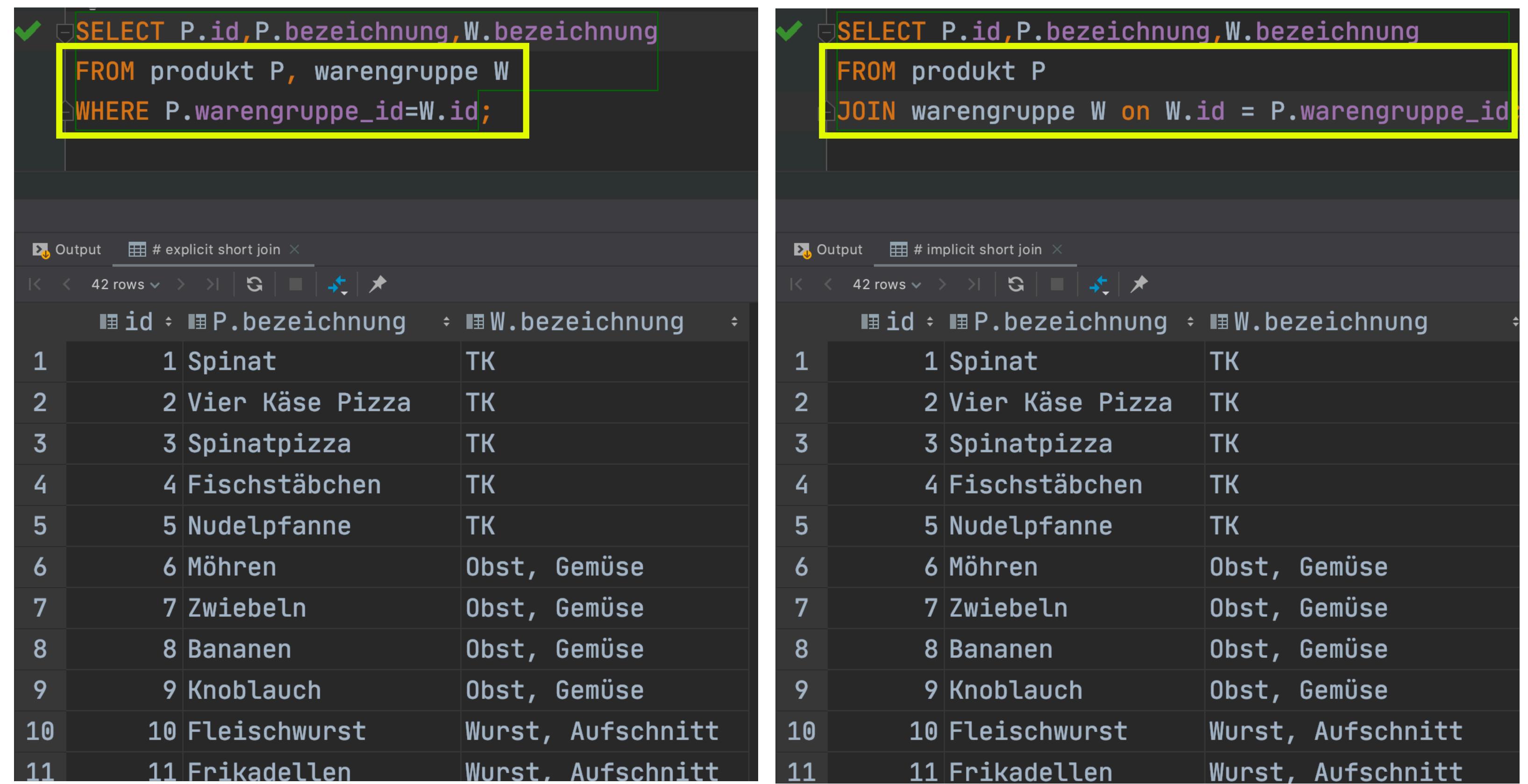
Beispiel matse\_mhist

# Motivation

---

## Warengruppe zu Produkt

- Wenn wir nach select nur die interessanten Attribute anzeigen, sind wir am Ziel.
- Die rechte Variante nutzt einen expliziten Befehl join und die on-Bedingung ist identisch der where-Bedingung.
- Und nun der Reihe nach...



The screenshot shows two database queries and their results. Both queries return the same data:

	P.id	P.bezeichnung	W.bezeichnung
1	1	Spinat	TK
2	2	Vier Käse Pizza	TK
3	3	Spinatpizza	TK
4	4	Fischstäbchen	TK
5	5	Nudelpfanne	TK
6	6	Möhren	Obst, Gemüse
7	7	Zwiebeln	Obst, Gemüse
8	8	Bananen	Obst, Gemüse
9	9	Knoblauch	Obst, Gemüse
10	10	Fleischwurst	Wurst, Aufschnitt
11	11	Frikadellen	Wurst, Aufschnitt

Beispiel matse\_mhist

## Cross Join

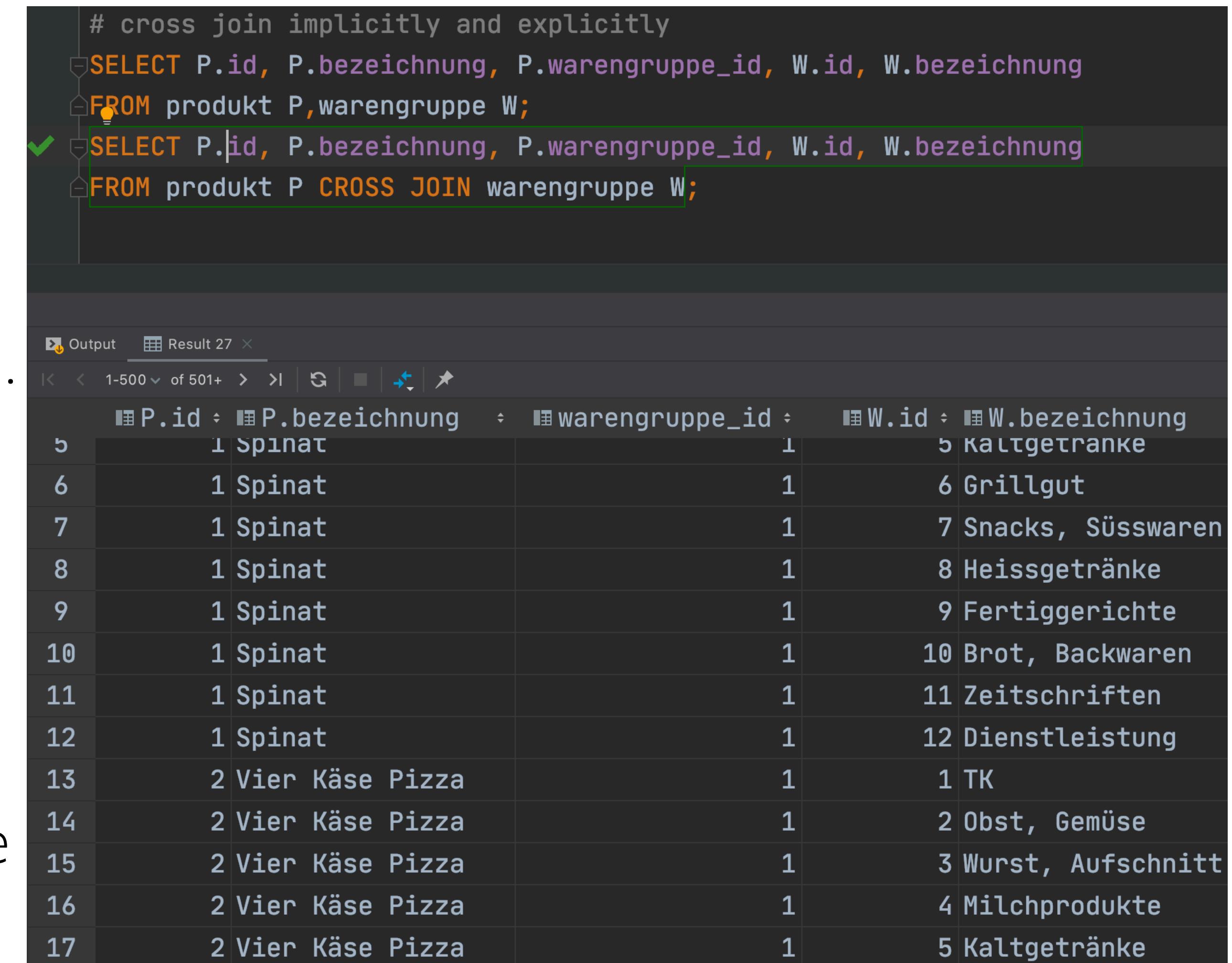
---

### Intention

- Die Angabe mehrerer Tabellen ohne weitere Einschränkungen führt zum Kreuzprodukt der Einträge - ein impliziter Cross Join (erstes select).
- Äquivalent dazu ist der explizite cross join (zweites select).

### Anmerkungen

- Jede Entität der ersten Tabelle wird mit jeder Entität der zweiten Tabelle kombiniert. Bei mehreren Tabellen entsprechend.



```
# cross join implicitly and explicitly
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P,warengruppe W;
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P CROSS JOIN warengruppe W;
```

P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
5	Spinat	1	5	Kaltgetränke
6	Spinat	1	6	Grillgut
7	Spinat	1	7	Snacks, Süßwaren
8	Spinat	1	8	Heissgetränke
9	Spinat	1	9	Fertiggerichte
10	Spinat	1	10	Brot, Backwaren
11	Spinat	1	11	Zeitschriften
12	Spinat	1	12	Dienstleistung
13	Vier Käse Pizza	1	1	TK
14	Vier Käse Pizza	1	2	Obst, Gemüse
15	Vier Käse Pizza	1	3	Wurst, Aufschnitt
16	Vier Käse Pizza	1	4	Milchprodukte
17	Vier Käse Pizza	1	5	Kaltgetränke

Beispiel matse\_mhist

## Inner Join

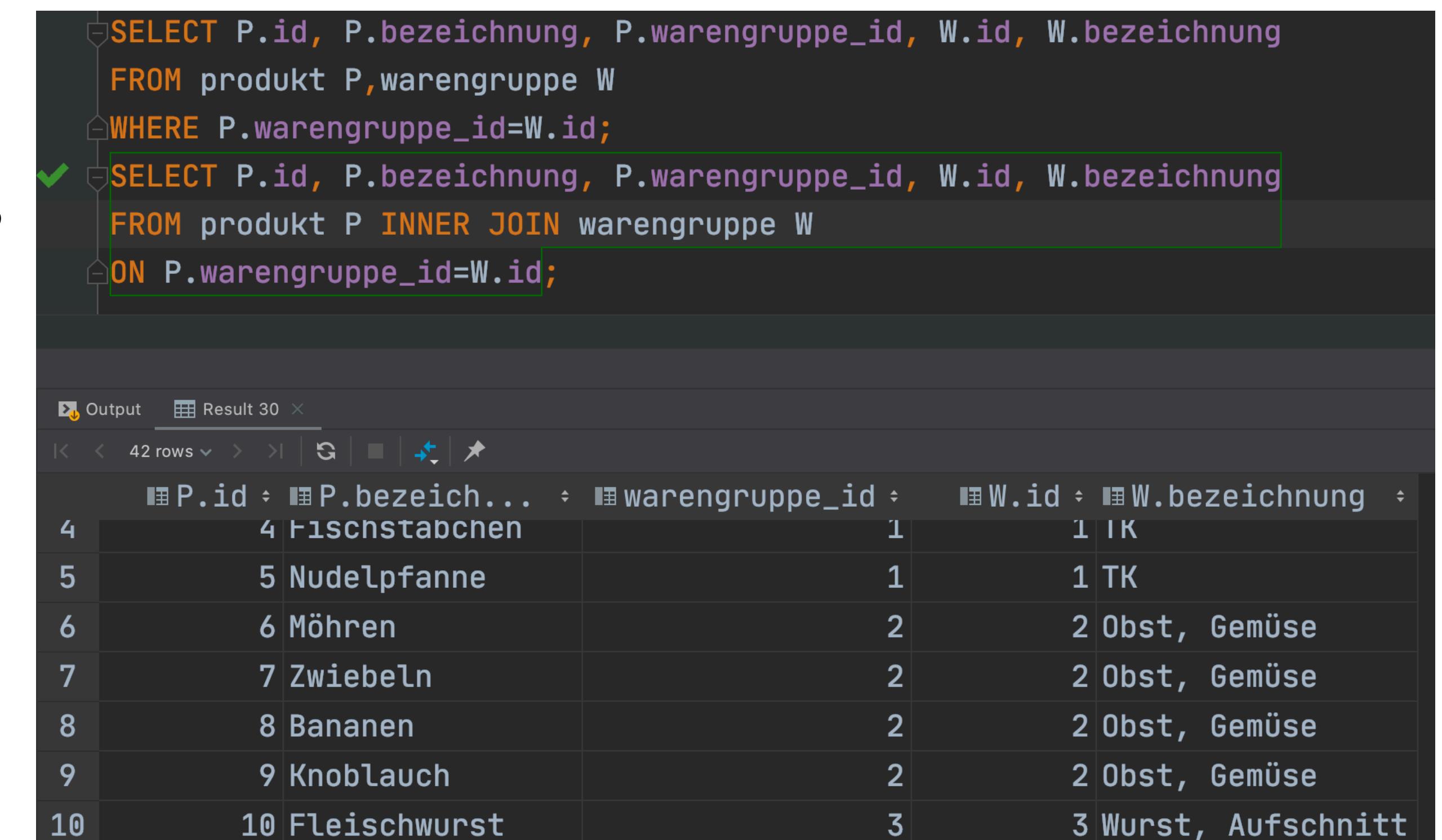
---

### Intention

- Wie zuvor geschildert, ist man oft nur an 'zusammengehörenden' Daten interessiert, also z.B. an der Bezeichnung der Warengruppe eines Produkts.
- Ein *Inner Join* ist als Selektion auf dem Kreuzprodukt definiert. Beide Joins hier sind äquivalent, allerdings gilt die obere implizite Form als veraltet.

### Anmerkungen

- Der explizite `inner join` erwarten die Angabe mittels `on`, welche Daten zusammengehören. '`inner`' ist hier optional.



The screenshot shows a MySQL query editor with two queries. The first query is the implicit form:

```
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P,warengruppe W
WHERE P.warengruppe_id=W.id;
```

The second query is the explicit form, highlighted with a green border:

```
SELECT P.id, P.bezeichnung, P.warengruppe_id, W.id, W.bezeichnung
FROM produkt P INNER JOIN warengruppe W
ON P.warengruppe_id=W.id;
```

Below the queries, the results are displayed in a table:

	P.id	P.bezeichnung	warengruppe_id	W.id	W.bezeichnung
4	4	Fischstäbchen	1	1	IK
5	5	Nudelpfanne	1	1	TK
6	6	Möhren	2	2	Obst, Gemüse
7	7	Zwiebeln	2	2	Obst, Gemüse
8	8	Bananen	2	2	Obst, Gemüse
9	9	Knoblauch	2	2	Obst, Gemüse
10	10	Fleischwurst	3	3	Wurst, Aufschnitt

## Inner Join

### Beispiele

- Alle Kunden, zu denen eine Bestellung existiert – und nur die!
- Alle Tiere und ihre 'Besitzer'. Achtung: Hier gibt es 'Rosa', die offenbar nicht Teil der Relation ist und die beim *Inner Join* auch nicht aufgeführt ist.

```
# example 1
SELECT B.id, B.kunde_id, K.id, K.name
FROM bestellung B INNER JOIN kunde K ON B.kunde_id=K.id;
```

Output # example 1

B.id	kunde_id	K.id	name
1	1	30	Sparkasse
2	2	30	Sparkasse
3	3	32	E.ON

```
# example 2
SELECT T.* FROM tier T;
SELECT T.name, M.name
FROM tier T JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

Output # example 2

T.id	T.name	Mitarbeiter_id
1	1 Petra	4
2	2 Mini	18
3	3 Rosa	<null>

```
# example 2
SELECT T.* FROM tier T;
SELECT T.name, M.name
FROM tier T JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

Output Result 39

T.name	M.name
Petra	Paul
Mini	Max

Beispiele matse\_mhist

## Unit 0x02 - Join

### Inner Join

#### Intention

- Joins über mehrere Tabellen abfragen.

```
# multiple inner joins
SELECT * FROM abteilung A;
SELECT * FROM standort S;
SELECT R.abteilung_id, R.st
FROM sitzt_am R;
```

```
# multiple inner joins
SELECT * FROM abteilung A;
SELECT * FROM standort S;
SELECT R.abteilung_id, R.st
FROM sitzt_am R;
```

```
# multiple inner joins
SELECT * FROM abteilung A;
SELECT * FROM standort S;
SELECT R.abteilung_id, R.standort_id
FROM sitzt_am R;
```

#### Anmerkungen

- N:M Beziehung: Abteilungen können auf mehrere Standorte aufgeteilt sein und Standorte können mehrere Abteilungen beherbergen.
- Die *Inner Joins* können einfach kombiniert werden. Die Definition bleibt bestehen.

```
SELECT R.abteilung_id,A.* ,R.standort_id,S.* FROM sitzt_am R
INNER JOIN abteilung A ON R.abteilung_id=A.id
INNER JOIN standort S ON R.standort_id=S.id;
```

abteilung_id	A.id	name	standort_id	S.id	ort
1	1	Vorstand	1	1	Aachen
2	2	HR/Buchhaltung	1	1	Aachen
3	3	Vertrieb	1	1	Aachen
6	6	F&E	1	1	Aachen
3	3	Vertrieb	2	2	Jülich
6	6	F&E	2	2	Jülich
3	3	Vertrieb	3	3	Köln

Beispiel matse\_mhist

### Inner Join

#### Beispiel

- Alle Mitarbeiter mit Abteilung und Funktion/Rolle dort.

```
SELECT R.* FROM arbeitet_in_als R;
SELECT M.id,M.name,A.name,L.beschreibung FROM arbeitet_in_als R
JOIN mitarbeiter M on M.id = R.mitarbeiter_id
JOIN abteilung A on A.id = R.abteilung_id
JOIN rolle L on L.id = R.rolle_id;
```

R

mitarbeiter_id	abteilung_id	rolle_id
12	4	4
13	4	4
23	4	8
14	5	3
14	5	4
15	5	4

```
SELECT R.* FROM arbeitet_in_als R;
SELECT M.id,M.name,A.name,L.beschreibung FROM arbeitet_in_als R
JOIN mitarbeiter M on M.id = R.mitarbeiter_id
JOIN abteilung A on A.id = R.abteilung_id
JOIN rolle L on L.id = R.rolle_id;
```

Result 60

M.id	M.name	A.name	beschreibung
4	Paul	AR	Mitarbeiter
5	Hannah	AR	Mitarbeiter
6	Luka	AR	Mitarbeiter
18	Max	F&E	Student
22	Lilly	F&E	Student
23	Tim	Marketing	Auszubildender

Beispiele matse\_mhist

## Outer Join

---

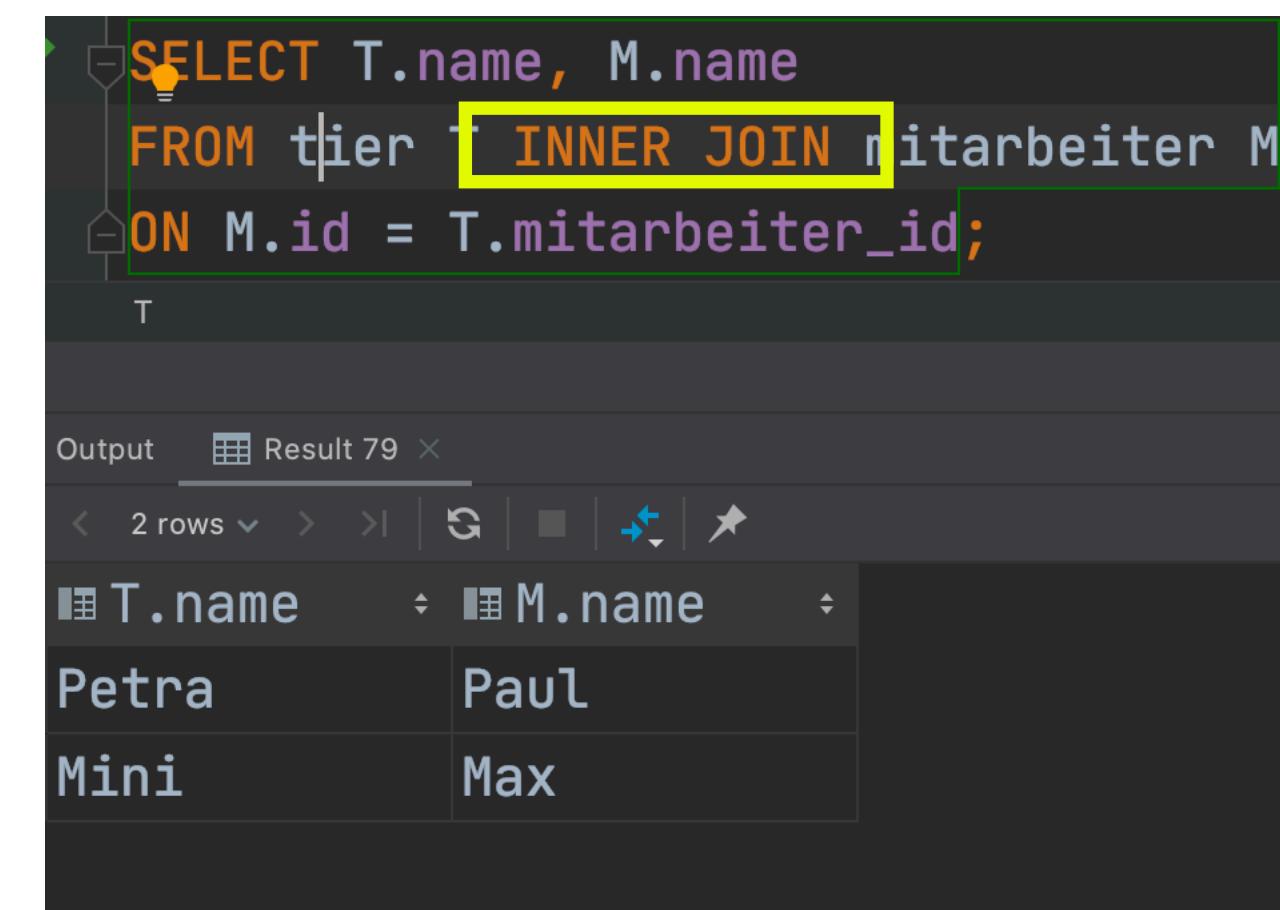
### Intention

- 'Zusammengehörige' Daten und Daten, zu denen kein Pendant existiert, abfragen. Das sind **Left Outer Join**, **Right Outer Join** und **Full Outer Join**.
- Bei einem *Left Outer Join* werden *alle* Entitäten der Entitätenmenge *links* der Relation berücksichtigt, auch wenn es keine zugehörigen Entitäten in der rechten Entitätenmenge gibt. Diese Attribute sind dann NULL.
- *Right Outer Join* analog, nur mit vertauschten Rollen.

### Beispiel

- Tiere mit und ohne 'Besitzer'.
- Wie schon gesehen, ist 'Rosa' beim *Inner Join* nicht dabei:

	<code>id</code>	<code>name</code>	<code>mitarbeiter_id</code>
1	Petra		4
2	Mini		18
3	Rosa		<null>



```

SELECT T.name, M.name
FROM tier T INNER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
    
```

The screenshot shows a MySQL command-line interface. A query is being typed in the command field. The 'ON M.id = T.mitarbeiter\_id' part of the 'INNER JOIN' clause is highlighted with a yellow box. Below the command, there are two tabs: 'Output' and 'Result 79'. The 'Output' tab shows the query and the 'Result' tab shows the results of the join. The results are as follows:

T.name	M.name
Petra	Paul
Mini	Max

Beispiele matse\_mhist

## Outer Join

### Beispiel Fortsetzung

- Beim *Left Outer Join* ist 'Rosa' dabei, da *alle* Entitäten des Typs Tier (links des Joins) berücksichtigt werden, auch wenn sie nicht in Relation stehen.  
Die zugehörigen Attribute des (nicht vorhandenen) Partners sind `NULL`.
- Betrachten wir das gleiche SQL-Kommando als *Right Outer Join*, so kommen *alle* Mitarbeiter (rechte Seite des Joins) vor und die zugehörigen Attribute sind `NULL`, wenn der Mitarbeiter nicht in Relation zu einem Tier steht.  
Hier ist 'Rosa' nicht dabei!

The screenshot shows a database interface with two queries and their results.

**Top Query (Left Outer Join):**

```
SELECT T.name, M.name
FROM tier T LEFT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

**Output:**

T.name	M.name
Petra	Paul
Mini	Max
Rosa	<null>

**Bottom Query (Right Outer Join):**

```
SELECT T.name, M.name
FROM tier T RIGHT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

**Output:**

T.name	M.name
Petra	Paul
Mini	Max
<null>	Mia
<null>	Ben
<null>	Emma
<null>	Hannah
<null>	Luke

Beispiele matse\_mhist

## Outer Join

### Beispiel Fortsetzung

- Der Full Outer Join ist die Vereinigung von *Left* und *Right Outer Join*. Das bedeutet, es sind alle Entitäten beider Seiten dabei, nur ggf. mit NULL-Einträgen.
- Manche DBMS haben keinen Befehl für einen *Full Outer Join*, dann vereinigt man die Ergebnisse der beiden *Outer Joins* einfach mit *union*.
- Vertauschen von Entitätstypen und *Left* und *Right Outer Joins*, ergibt wieder den ersten Outer Join.

```
SELECT T.name, M.name
FROM mitarbeiter M RIGHT OUTER JOIN tier T
ON M.id = T.mitarbeiter_id;
```

T.name	M.name
Petra	Paul
Mini	Max
Rosa	<null>

```
SELECT T.name, M.name
FROM tier T LEFT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id
UNION
SELECT T.name, M.name
FROM tier T RIGHT OUTER JOIN mitarbeiter M
ON M.id = T.mitarbeiter_id;
```

T.name	M.name
Petra	Paul
Mini	Max
Rosa	<null>
<null>	Mia
<null>	Ben
<null>	Emma
<null>	Hannah

Beispiele matse\_mhist

## Outer Join

### Beispiel

- Nur die Kunden, zu denen eine Bestellung existiert, d.h. die in Relation stehen – somit *Inner Join*:
- Kunden, zu denen eine Bestellung existiert und die, die noch keine Bestellung aufgegeben haben, d.h. faktisch *alle* Kunden, mit und ohne Partner – somit *Outer Join*:
- Der Entitätstyp *kunde* steht *rechts* am *Join*, daher hier *Right Outer Join*.
- Da beim *Inner Join* nur Entitäten teilnehmen, die einen Partner haben, gibt es keinen *Left* oder *Right Inner Join*.

```
SELECT B.id, B.kunde_id, K.id, K.name
FROM bestellung B INNER JOIN kunde K
ON B.kunde_id=K.id;
```

Output # example

B.id	kunde_id	K.id	name
1	30	30	Sparkasse
2	30	30	Sparkasse
3	32	32	E.ON

```
SELECT B.id, B.kunde_id, K.id, K.name
FROM bestellung B RIGHT OUTER JOIN kunde K
ON B.kunde_id=K.id;
```

Output # example

B.id	kunde_id	K.id	name
<null>	<null>	29	Deutsche Bank
1	30	30	Sparkasse
2	30	30	Sparkasse
<null>	<null>	31	Börse
3	32	32	E.ON
<null>	<null>	33	Infinion

Beispiele matse\_mhist

## Natural Join

### Intention

- Bei Tabellen mit gleichnamigen Attributen, z.B. bei Fremdschlüsselbeziehungen, kann auf die explizite Angabe der Bedingung (mit `on`, `using` oder `where`) verzichtet werden. Das DBMS verwendet dann *alle* gleichnamigen Attribute - ein sog. Natural Join.

### Beispiel

- Studierende hören Vorlesungen, einmal mit `natural join` und einmal mit `join/on`.

Table R:

```
SELECT R.* FROM hoeren R;
```

	R
Output	# natural join
< 15 rows >   >   ⌂   +   -   Tx: Auto   DB ↕	
<b>MatrNr</b>   <b>VorlNr</b>	
25403   5022	
26120   5001	
27550   4052	

Table S:

```
SELECT S.* FROM studenten S;
```

	S
Output	kemper.studenten
< 8 rows >   >   ⌂   +   -   Tx: Auto   DB ↕	
<b>MatrNr</b>   <b>Name</b>   <b>Semester</b>	
24002   Xenokrates   18	
25403   Jonas   12	
26120   Fichte   10	

Table V:

```
SELECT V.* FROM Vorlesungen V;
```

	V
Output	Result 92
< 10 rows >   >   ⌂   +   -   Tx: Auto   DB ↕	
<b>VorlNr</b>   <b>Titel</b>	
4052   Logik	
4630   Die 3 Kritiken	
5001   Grundzuege	

Query 1 (NATURAL JOIN):

```
SELECT S.Name, V.Titel
FROM Studenten S
NATURAL JOIN hoeren R
NATURAL JOIN Vorlesungen V;
```

Query 2 (JOIN ON):

```
SELECT S.Name, V.Titel
FROM Studenten S
JOIN hoeren R ON S.MatrNr = R.MatrNr
JOIN Vorlesungen V ON R.VorlNr = V.VorlNr;
```

Result 95 (Query 1):

Name	Titel
Jonas	Glaube und Wissen
Fichte	Grundzuege

Result 96 (Query 2):

Name	Titel
Jonas	Glaube und Wissen
Fichte	Grundzuege

Beispiele kemper

## Natural Join

---

### Anmerkungen

- Von der Verwendung von *Natural Joins* wird abgeraten!
- Das DBMS verwendet immer *alle* gleichnamigen Attribute, d.h. durch Hinzufügen neuer Attribute oder Umbenennen vorhandener Attribute, von denen eins zufällig so heisst wie in einem per *Natural Join* eingebunden anderen Entitätstyp, führt zu einer Änderung der Abfrage!
- Erweitern einer Tabelle ist nichts ungewöhnliches und passiert ggf. sogar automatisiert, wenn z.B. die zu persistierende Objektstruktur erweitert wird und die Tabellen die Daten widerspiegeln. Das passiert schnell ohne mögliche *Natural Joins* zu berücksichtigen - und dann funktionieren die Abfragen nicht mehr.

## Self Join

---

### Intention

- Tabellen, die selber Fremdschlüsselbeziehung enthalten, abfragen: ein *Self Join*.

### Beispiel

- In `mitarbeiter` ist der jeweilige Vorgesetzte als Fremdschlüssel in der Tabelle abgelegt.
- Es ist ein klassischer *Inner Join*, nur steht der Entitätstyp zu sich selbst in Relation.

The screenshot shows a MySQL command-line interface. The SQL query is:

```
SELECT M.id, M.name, N.name 'Vorgesetzter'  
FROM mitarbeiter M  
INNER JOIN mitarbeiter N  
ON M.vorgesetzter_mitarbeiter_id=N.id;
```

The output shows a table with three columns: id, name, and 'Vorgesetzter'. The data is:

	id	name	'Vorgesetzter'
1	Mia	GF	
2	Ben	GF	
3	Emma	Leon	
4	Paul	AR	
5	Hannah	AR	
6	Luka	AR	
7	Sofia	Mia	

Beispiel matse\_mhist

## Check

---

### Aufgaben

Die benötigten Tabellen entstammen dem `matse_mhist` Schema.

1. Geben Sie für alle Bestellungen (`bestellung`) jeweils das Datum und den Namen des Kunden (`kunde`) aus.

(a) Nutzen Sie einmal einen (*Inner*) *Join* mit `where`, und

(b) einmal einen *Join* mit `join`.

(c) Jetzt fügen Sie noch den Namen des vermittelnden Mitarbeiters (`mitarbeiter`) hinzu.

Datum	Kunde
2014-09-12 00:00:00	Sparkasse
2014-08-30 00:00:00	Sparkasse
2014-09-11 00:00:00	E.ON

Ergebnis (a) und (b)

Datum	Kunde	vermittelt
2014-09-12 00:00:00	Sparkasse	Anna
2014-08-30 00:00:00	Sparkasse	Finn
2014-09-11 00:00:00	E.ON	Anna

Ergebnis (c)

## Check

---

### Aufgaben

2. Geben Sie zu jeder Abteilung (abteilung) den Standort (standort) aus (sitzt\_am).
- Nutzen Sie einen (*Inner*) *Join* mit *join*.
  - Geben Sie auch Abteilungen aus, die (noch) keinen Standort haben.
  - Formulieren Sie (b) einmal als *Left* und einmal als *Right Outer Join*.
  - Beantworten Sie mit einem SQL-Befehl die Frage, welche Abteilung bislang keinen Standort besitzt. Nutzen Sie die Funktion *isnull* (*attribut*).

	name	ort
1	Vorstand	Aachen
2	HR/Buchhaltung	Aachen
3	Vertrieb	Aachen
4	F&E	Aachen
5	Vertrieb	Jülich
6	F&E	Jülich
7	Vertrieb	Köln
8	Einkauf	Köln
9	Marketing	Köln
10	Vertrieb	Berlin

Ergebnis (a)

	name	ort
1	Vorstand	Aachen
2	HR/Buchhaltung	Aachen
3	Vertrieb	Jülich
4	Vertrieb	Aachen
5	Vertrieb	Köln
6	Vertrieb	Berlin
7	Marketing	Köln
8	Einkauf	Köln
9	F&E	Aachen
10	F&E	Jülich
11	AR	<null>

Ergebnis (b) und (c)

	name	ort
1	AR	<null>

Ergebnis (d)

## Check

---

### Aufgaben

3. Nun geht es um Mitarbeiter (mitarbeiter), Vorgesetzte (Fremdschlüssel vorgesetzter\_mitarbeiter\_id in mitarbeiter), Abteilungen (abteilung) und Rollen (rolle).

(a) Geben Sie alle Mitarbeiter aus, deren Vorgesetzte 'Mia' ist.

(b) Geben Sie alle Mitarbeiter mit Namen, der Abteilung und Rolle sowie den jeweiligen Namen der oder des Vorgesetzten aus.

Tipp: Starten Sie mit der Relation arbeitet\_in\_als.

	■ name	■ Vorgesetzte
1	Sofia	Mia
2	Jonas	Mia
3	Leon	Mia
4	Luis	Mia
5	Lukas	Mia
6	Leonie	Mia

Ergebnis (a)

	■ M1.name	■ A.name	■ beschreibung	■ Vorgesetzte
1	Mia	Vorstand	Leitung	GF
2	Mia	HR/Buchhaltung	Leitung	GF
3	Ben	Vertrieb	Leitung	GF
4	Leon	Marketing	Leitung	Mia
5	Luis	Einkauf	Leitung	Mia
6	Lukas	F&E	Leitung	Mia
7	Paul	AR	Leitung	AR
8	Mia	Vorstand	Mitarbeiter	GF
9	Ben	Vorstand	Mitarbeiter	GF
10	Emma	Vorstand	Mitarbeiter	Leon

Ergebnis (b)

## Check

---

### Aufgaben

4. Analysieren Sie ein Kundennetzwerk (kunde und kennt).

Beachten Sie, dass die Kenntnis 'Kunde a' kennt 'Kunde b' für diese Aufgabe weder kommutativ noch transitiv ist.

- (a) Geben Sie die Namen aller Verbindungen der Banken und Sparkassen (Kunde a hat 'bank' oder 'kasse' im Namen) im Kundennetzwerk aus.

- (b) Geben Sie alle Kunden (kunde a) aus, die noch keine Verbindungen geknüpft haben.

kunde_a_id	kunde_b_id
2	31
34	31
30	31
29	30
27	28

kennt

K1.name	K2.name
Sparkasse	Börse
Deutsche Bank	Sparkasse
Commerzbank	Deutsche Bank
Commerzbank	Sparkasse

Ergebnis (a)

name
Börse
Infinion

Ergebnis (b)

# UNIT 0x03

# GROUP FUNCTIONS

## Group Functions

---

### Lernziel

- Datenaggregation und Daten nach Kriterien gruppieren.  
→ Group Functions bzw. Aggregierungsfunktionen.

# Group Functions

---

## Intention

- Group Functions verwenden.

## Beispiel

- Hier werden über alle Produkte die angegebenen Aggregationsfunktionen berechnet, d.h. Minimum, Maximum, Summe, Anzahl, Durchschnitt, etc.
- stddev\_pop bezieht sich auf die sog. 'Population standard deviation', im Gegensatz ZUR stddev\_samp 'Sample standard deviation'.

SELECT \* FROM produkt;

	# all data	Tx: Auto	DB	DML
1	Spinat	1.99		
id	bezeichnung	stueckpreis		
2	Vier Käse Pizza	2.39		
3	Spinatpizza	2.29		
4	Fischstäbchen	1.99		
5	Nudelpfanne	3.29		

Beispiel matse\_mhist

```
SELECT min(stueckpreis), max(stueckpreis), sum(stueckpreis), count(stueckpreis),
       sum(stueckpreis)/count(stueckpreis), avg(stueckpreis), stddev_pop(stueckpreis),
       sqrt(var_pop(stueckpreis)) FROM produkt;
```

	min...	max...	sum...	count...	sum(stue...	avg...	stddev...	sqrt(var_p...
1	0.29	512.00	725.55	42	17.275000	17.275000	79.616306	79.616...

## Group Functions

---

### Intention

- Group Functions auf Selektionen verwenden.

### Anmerkung

- Bei der Verwendung spielt eine Rolle, ob man NULL Werte und/oder doppelte Werte berücksichtigt.

### Beispiele

- Hier wird jeweils der Durchschnittspreis aller Produkte *nur der Warengruppe 1* berechnet.

id	bezeichnung	stueckpreis
1	Spinat	1.99
2	Vier Käse Pizza	2.39
3	Spinatpizza	2.29
4	Fischstäbchen	1.99
5	Nudelpfanne	3.29

Beispiel matse\_mhist

# Group Functions

---

## Beispiele

- count (\*) zählt alle Zeilen inkl. NULL, (kommt hier allerdings nicht vor),
- count (stueckpreis) ohne NULL und
- count (distinct stueckpreis) nur verschiedene Preise.

```
SELECT count(*), warengruppe_id, avg(stueckpreis) FROM produkt
WHERE warengruppe_id=1;
   `count(*)`  `warengruppe_id`  `avg(stueckpreis)`
1      5           1        2.390000
```

```
SELECT count(stueckpreis), warengruppe_id, avg(stueckpreis) FROM produkt
WHERE warengruppe_id=1;
   `count(stueckpreis)`  `warengruppe_id`  `avg(stueckpreis)`
1          5           1        2.390000
```

```
SELECT count(distinct stueckpreis), warengruppe_id, avg(stueckpreis) FROM produkt
WHERE warengruppe_id=1;
   `count(distinct stueckpreis)`  `warengruppe_id`  `avg(stueckpreis)`
1            4           1        2.390000
```

```
SELECT count(distinct stueckpreis), warengruppe_id, avg(distinct stueckpreis) FROM produkt
WHERE warengruppe_id=1;
   `count(distinct stueckpreis)`  `warengruppe_id`  `avg(distinct stueckpreis)`
1            4           1        2.490000
```

Beispiel matse\_mhist

# Group Functions

---

## Intention

- Group Functions auf *gruppierten* Daten verwenden.

## Beispiel

- Bedingung group by teilt die Gesamtmenge in Teilmengen (Gruppierung), über die jeweils der Stückpreis ermittelt wird.
- Vgl. count und avg vorheriges Beispiel.

```
SELECT count(*), warengruppe_id,
       min(stueckpreis), max(stueckpreis), avg(stueckpreis) FROM produkt
GROUP BY warengruppe_id;
```

Output # Group Functions au...rten Daten verwenden. ×

	count(*)	warengruppe_id	min...	max...	avg...
1	5		1	1.99	3.29
2	4		2	0.29	1.29
3	2		3	1.99	2.35
4	3		4	0.79	0.99
5	4		5	0.99	1.80
6	3		6	2.79	3.55
7	3		7	1.59	1.99
8	2		8	2.50	3.20
9	4		9	1.89	2.39
10	4		10	1.90	2.80
11	6		11	0.90	6.50
12	2		12	128.00	512.00
					320.000000

Beispiel matse\_mhist

## Group Functions

---

### Intention

- Group Functions auf *gruppierten Daten einer Selektion* verwenden.

### Beispiel

- Bedingung `group by` teilt die mit `where` selektierte Menge in Teilmengen (Gruppierung), über die jeweils der Stückpreis ermittelt wird.
- Wie zuvor, nur dass die Gesamtmenge, die betrachtet wird, vorher gefiltert wird.

```
min(stueckpreis), max(stueckpreis), avg(stueckpreis)
FROM produkt WHERE warengruppe_id IN (1,2,4)
GROUP BY warengruppe_id;
```

Output # Group Functions au... Selektion verwenden.

	count...	warengruppe_id	min...	max...	avg...
1	5	1	1.99	3.29	2.390000
2	4	2	0.29	1.29	0.737500
3	3	4	0.79	0.99	0.920000

Beispiel matse\_mhist

## Group Functions

### Intention

- Group Functions auf gruppierten Daten *mit Bedingung* verwenden.

### Beispiel

- Bedingung having filtert die mit group by gruppierten Teilmengen.
- Wie zuvor, nur dass diesmal die Bedingung auf das *Ergebnis* der Gruppierung angewendet wird, vgl. min Beispiel zuvor.
- Wichtig: Hier kommt es immer wieder zu Verwechslungen...

```
SELECT count(*), warengruppe_id,
       min(stueckpreis), max(stueckpreis), avg(stueckpreis)
  FROM produkt WHERE warengruppe_id IN (1,2,4)
 GROUP BY warengruppe_id HAVING min(stueckpreis)>1;
```

	count(*)	warengruppe_id	min...	max...	avg...
1	5	1	1.99	3.29	2.390000

Beispiel matse\_mhist

# Group Functions

## Intention

- Group Functions korrekt verwenden.

## Anmerkung

- Sowohl `bezeichnung` als auch `einheit` sind *innerhalb* einer Warengruppe unterschiedlich!  
Wenn jetzt bezüglich der Warengruppe gruppiert wird, dann ist zwar *technisch* die Angabe `bezeichnung` und `einheit` im select möglich, aber **nicht sinnvoll** und in manchem DBMS auch **verboten**.

```
SELECT bezeichnung, warengruppe_id, stueckpreis, einheit FROM produkt
WHERE warengruppe_id IN (1,2,4);
```

Output # Group Functions korrekt verwenden. ×

bezeichnung	warengruppe_id	stueckpreis	einheit
Spinat	1	1.99	PK
Vier Käse Pizza	1	2.39	ST
Spinatpizza	1	2.29	ST
Fischstäbchen	1	1.99	PK
Nudelpfanne	1	3.29	PK
Möhren	2	0.39	KG
Zwiebeln	2	0.29	KG

```
SELECT bezeichnung, warengruppe_id, min(stueckpreis), einheit FROM produkt
GROUP BY warengruppe_id;
```

Output Result 32 ×

bezeichnung	warengruppe_id	`min(stueckpreis)`	einheit
Spinat	1	1.99	PK
Möhren	2	0.29	KG
Fleischwurst	3	1.99	ST
Milch	4	0.79	ST
Cola light	5	0.99	ST

Beispiel matse\_mhist

## Group Functions

---

### Intention

- Group Functions *mit Alias* verwenden.

### Beispiel

- Hier wird ein Alias s für das Minimum angegeben und in having verwendet.
- s kann genauso wie min nicht in where verwendet werden, da der Wert da noch nicht feststeht.
- Zur Sortierung wiederum ist s geeignet.

```
SELECT count(*), warengruppe_id,
       min(stueckpreis) S FROM produkt
WHERE warengruppe_id IN (1,2,4)
GROUP BY warengruppe_id HAVING 3*S>1
ORDER BY S;
```

Output # Group Functions mit Alias verwenden. ×

count(*)	warengruppe_id	S
3	4	0.79
5	1	1.99

Beispiel matse\_mhist

# Group Functions

## Intention

- group by mit mehreren Attributen verwenden.

## Beispiel

- Obere Tabelle zeigt zum Vergleich die Anzahl (#) Produkte je einheit und umsatzsteuer.
- Gruppiert wird zunächst nach umsatzsteuer und innerhalb dieser Gruppe nach einheit.

```
SELECT count(*) FROM produkt
WHERE einheit='KG' AND umsatzsteuer=0.07; # 4
SELECT count(*) FROM produkt
WHERE einheit='PK' AND umsatzsteuer=0.07; # 14
SELECT count(*) FROM produkt
WHERE einheit='ST' AND umsatzsteuer=0.07; # 16
# ...
SELECT count(*) FROM produkt
WHERE einheit='ST' AND umsatzsteuer=0.19; # 6
```

Output Result 43 ×

```
SELECT count(umsatzsteuer), umsatzsteuer,
       count(einheit), einheit,
       min(stueckpreis), max(stueckpreis)
  FROM produkt GROUP BY umsatzsteuer, einheit
 ORDER BY umsatzsteuer;
```

count...	umsatzsteuer	count...	einheit	min...	max...
4	0.07	4	KG	0.29	1.29
14	0.07	14	PK	1.59	3.55
16	0.07	16	ST	0.79	6.50
1	0.19	1	1x	128.00	128.00
1	0.19	1	nx	512.00	512.00
6	0.19	6	ST	0.99	3.20

Beispiel matse\_mhist

## Group Functions

---

### Intention

- Group Functions mit Joins verwenden.

### Beispiel

- Obere Tabelle zeigt eine Übersicht aller 'Getränke'. Untere Tabelle zeigt diese nach Warenguppe gruppiert.
- Strenggenommen dürfte `W.bezeichnung` nicht im `select` auftauchen, aber dieses Attribut ist natürlich innerhalb der Warenguppe gleich.

```
SELECT P.warengruppe_id, P.stueckpreis, W.bezeichnung
FROM produkt P
INNER JOIN warengruppe W ON P.warengruppe_id=W.id
WHERE W.bezeichnung LIKE '%getränke';
```

Output # Group Functions mit Joins verwenden. ×

6 rows > >> G ⚡ ⚡

warengruppe_id	stueckpreis	bezeichnung
5	1.50	Kaltgetränke
5	1.50	Kaltgetränke
5	1.80	Kaltgetränke
5	0.99	Kaltgetränke
8	2.50	Heissgetränke
8	3.20	Heissgetränke

```
SELECT count(P.stueckpreis), P.warengruppe_id, avg(P.stueckpreis), W.bezeichnung
FROM produkt P
INNER JOIN warengruppe W ON P.warengruppe_id=W.id
WHERE W.bezeichnung LIKE '%getränke' GROUP BY warengruppe_id;
```

Output Result 58 ×

2 rows > >> G ⚡ ⚡

`count(P.stueckpreis)`	warengruppe_id	`avg(P.stueckpreis)`	bezeichnung
4	5	1.447500	Kaltgetränke
2	8	2.850000	Heissgetränke

Beispiel matse\_mhist

## Group Functions

---

### Übersicht

- `where` selektiert die *Gesamtmenge*, auf der danach gruppiert werden kann.
- `group by` gruppiert und wendet ggf. Aggregationsfunktionen an.  
Achtung: Diese können in der `where` Bedingung nicht verwendet werden.
- `having` ist eine Bedingung an die *Ergebnisse der Gruppierung*, hier kann das Schlüsselwort `where` nicht verwendet werden.
- Selektierte Attribute müssen in der `group by` Bedingung vorkommen, da sonst deren Wert innerhalb der Gruppierung nicht eindeutig und nicht sinnvoll ist.
- Ausdrücke können mit einem Alias versehen werden, der in `having` benutzt werden kann.
- Gruppierung kann auch *Joins* umfassen.

## Check

---

### Aufgaben

Die benötigten Tabellen entstammen dem `matse_mhist` Schema.

1. Gucken Sie sich zunächst die Gehaltsstruktur der Mitarbeiter mit einem echten Jahresgehalt ( $>0$ ) an und geben Sie sie nach der `id` des Vorgesetzten geordnet aus.

<code>id</code>	<code>name</code>	<code>jahresgehalt</code>	<code>vorgesetzter_mitarbeiter_id</code>	<code>Boss</code>
12	Leon	70000.00		1 Mia
20	Leonie	70000.00		1 Mia
16	Lukas	70000.00		1 Mia

(a) Gruppieren Sie nun die echten Jahresgehälter nach der `id` des Vorgesetzten und ermitteln Sie das min. und das max. Gehalt sowie die Anzahl von Gehältern innerhalb der jeweiligen Gruppe. Ordnen Sie sie und überprüfen Sie die Plausibilität des Ergebnisses.

<code>name</code>	<code>`count...`</code>	<code>`min...`</code>	<code>`max...`</code>
Mia	6	50000.00	70000.00
Ben	3	50000.00	50000.00
Leon	3	100000.00	70000.00

(b) Ergänzen Sie die Ausgabe um das Durchschnittsgehalt und geben Sie nur die Einträge aus, bei denen dieses über 50000 liegt.

<code>name</code>	<code>`count...`</code>	<code>`min...`</code>	<code>`max...`</code>	<code>avg</code>
Mia	6	50000.00	70000.00	63333.333333
GF	2	90000.00	110000.00	100000.000000

## Check

---

### Aufgaben

2. Fragen zum Produktsortiment. Stellen Sie Ihre Antworten analog zusammen.

(a) Was kostet eine Pizza im Produktsortiment im Durchschnitt und wie viele Produkte gibt es davon?

pizzen	avg
2	2.340000

(b) Wie viele Produkte gibt es in den Warengruppen 1 und 2?

bezeichnung	count(P.id)
TK	5
Obst, Gemüse	4

(c) Ermitteln die Anzahl Produkte je Warengruppe, deren min. Stückpreis über 2€ liegt.

bezeichnung	count...	min
Grillgut	3	2.79
Heissgetränke	2	2.50
Dienstleistung	2	128.00

(d) Wie viele Produkte gibt es im Preissegment zwischen 1€ und 2€ je Umsatzsteuersatz?

`count...	umsatzsteuer
12	0.07
3	0.19

## Check

---

### Aufgaben

3. Sehen Sie sich zunächst die Relation `arbeitet_in_als` im DBMS geeignet an. Ein Mitarbeiter kann in mehreren Abteilungen in verschiedenen Rollen arbeiten.

Welche Mitarbeiter arbeiten *in mehr als zwei Rollen* in einer Abteilung (siehe Beispiel 'Lilly')?

*Tipp:* Starten Sie zunächst mit einfach Abfragen und Gruppierungen, ggf. ohne *Joins*, und erhöhen Sie nach und nach die Komplexität. Gucken Sie sich ausgewählte Beispiele (etwa 'Lilly') aus Plausibilitätsgründen an.

M.name	A.name	beschreibung
Leon	Marketing	Leitung
Leonie	Marketing	Mitarbeiter
Lilly	F&E	Mitarbeiter
Lilly	F&E	Student
Lilly	F&E	Mitarbeiter
Lilly	F&E	Auszubildender
Luis	Einkauf	Mitarbeiter
Luis	Einkauf	Leitung
Luka	AR	Mitarbeiter

M.name	A.name	rollen
Max	F&E	3
Lilly	F&E	3

# UNIT 0x04

## SUBSELECTS

## Subselects

---

### Lernziel

- Unterabfragen benutzen  
→ *Subselects*.

## Einwertige Subselects

### Intention

- Abfragen mit Bedingungen bzw. Operanden, die unbekannt sind und mit eigenen Unterabfragen, den subselects, erst ermittelt werden müssen.

### Beispiel

- Alle Mitarbeiter suchen, deren Vorgesetzte 'Mia' ist.
- Problem: Die id von 'Mia' ist nicht bekannt, daher muss diese zunächst ermittelt werden. Die absolute Angabe einer Konstanten ist jedoch fehleranfällig.
- Wir nutzen stattdessen ein Subselect. Dessen Ergebnis ist *ein Wert* und der fehlende Operand.

```
SELECT M.id, M.name FROM mitarbeiter M  
WHERE M.vorgesetzter_mitarbeiter_id = 1;
```

Beispiel matse\_mhist, so nicht!

```
SELECT M.id, M.name FROM mitarbeiter M  
WHERE M.vorgesetzter_mitarbeiter_id = (  
    SELECT id FROM mitarbeiter WHERE name='Mia'  
)
```

	id	name
1	7	Sofia
2	8	Jonas
3	12	Leon

Beispiel matse\_mhist

## Einwertige Subselects

---

### Beispiele

- Finde alle Mitarbeiter, die ein höheres Jahresgehalt als 'Luis' bekommen.
- Finde alle Mitarbeiter von 'Mia', die ein höheres Gehalt als 'Sofie' bekommen.

```
SELECT M.id, M.name FROM mitarbeiter M
WHERE M.jahresgehalt > (
    SELECT jahresgehalt FROM mitarbeiter WHERE name='Luis'
);
```

1	1	Mia
2	2	Ben

Beispiel matse\_mhist

```
SELECT M.id, M.name FROM mitarbeiter M
WHERE M.jahresgehalt > (
    SELECT jahresgehalt FROM mitarbeiter WHERE name='Sofia'
) and M.vorgesetzter_mitarbeiter_id = (
    SELECT id FROM mitarbeiter WHERE name='Mia'
);
```

1	12	Leon
2	14	Luis
3	16	Lukas

## Einwertige Subselects

---

### Intention

- Es können auch Group Functions in Subselects verwendet werden, ebenso Subselects in HAVING.

### Beispiel

- Es werden alle Mitarbeiter gefunden, die das Minimalgehalt erhalten.

```
SELECT M.id, M.name, jahresgehalt FROM mitarbeiter M
WHERE M.jahresgehalt = (
    SELECT min(jahresgehalt) FROM mitarbeiter
    WHERE jahresgehalt > 0
);
```

M.id	M.name	jahresgehalt
1	4 Paul	5000.00
2	5 Hannah	5000.00
3	6 Luka	5000.00

Beispiel matse\_mhist

## Subselects und IN

---

### Intention

- Bislang ermittelten die Subselects genau ein Ergebnis, oder anders gesagt, eine Ergebniszeile. Es können auch Ergebnisse bestehend aus mehreren Zeilen und mehreren Attributen mit den Operatoren IN, ANY und ALL verwendet werden.

### Beispiele IN

- Es werden alle Mitarbeiter gefunden, die ein Gehalt von 10000, 12000 oder 14000 erhalten.
- Es werden alle Mitarbeiter gefunden, die ein Minimalgehalt aus der Menge aller Minimalgehälter gruppiert nach Vorgesetzten erhalten.

Beispiel matse\_mhist

```
SELECT name, jahresgehalt FROM mitarbeiter  
WHERE jahresgehalt IN (10000,12000,14000);
```

```
SELECT name, jahresgehalt FROM mitarbeiter  
WHERE jahresgehalt IN (  
    SELECT min(M.jahresgehalt) FROM mitarbeiter M  
    GROUP BY M.vorgesetzter_mitarbeiter_id  
)
```

	name	jahresgehalt
1	Sofia	50000.00
2	Jonas	50000.00

## Subselects und ANY

### Beispiele ANY

- <ANY ergibt alle Produkte, deren Preis kleiner als irgendein Preis der Produkte der Warengruppe 4 ist, d.h. kleiner als das Maximum; siehe Beispiel.
- >ANY ergibt alle Produkte, deren Preis größer als irgendein Preis der Produkte der Warengruppe 4 ist, d.h. größer als das Minimum.
- =ANY liefert alle Produkte, deren Preis mit irgendeinem Preis der Produkte der Warengruppe 4 übereinstimmt - das entspricht IN.

```
SELECT bezeichnung, stueckpreis, warengruppe_id
FROM produkt P WHERE P.stueckpreis < ANY (
    SELECT stueckpreis FROM produkt WHERE warengruppe_id=4
) AND P.warengruppe_id <> 4;
```

bezeichnung	stueckpreis	warengruppe_id
Knoblauch	0.98	2
Bild	0.90	11
Möhren	0.39	2
Zwiebeln	0.29	2

Beispiel matse\_mhist

## Subselects und ALL

### Beispiele ALL

- <ALL ergibt alle Produkte, deren Preis kleiner als alle Preise der Produkte der Warengruppe 4 ist, d.h. kleiner als das Minimum; siehe Beispiel.
- >ALL ergibt analog alle Produkte, deren Preis größer als alle Preise der Produkte der Warengruppe 4 ist, d.h. größer als das Maximum.
- NOT kann mit den Operatoren IN, ANY und ALL verwendet werden.

```
SELECT bezeichnung, stueckpreis, warengruppe_id
FROM produkt P WHERE P.stueckpreis < ALL (
    SELECT stueckpreis FROM produkt WHERE warengruppe_id=4
) AND P.warengruppe_id >> 4;
```

	bezeichnung	stueckpreis	warengruppe_id
1	Möhren	0.39	2
2	Zwiebeln	0.29	2

Beispiel matse\_mhist

## Mehrwertige Subselects

---

### Intention

- Ebenfalls möglich sind Abfragen, in denen mehrere Attribute im Subselect verwendet werden.

### Beispiel

- Es werden alle Produkte gefunden, die einen Preis *und* eine Einheit wie mind. ein Produkt aus der Warengruppe 1 besitzen.
- Es werden nur Einträge gefunden, bei denen *beide* Attribute übereinstimmen.

```
SELECT bezeichnung, einheit, stueckpreis, warengruppe_id FROM produkt
WHERE (einheit, stueckpreis) in (
    SELECT einheit, stueckpreis FROM produkt
    WHERE warengruppe_id=1
) and warengruppe_id <> 1;
```

bezeichnung	einheit	stueckpreis	warengruppe_id
1 Chips	PK	1.99	7
2 Nudeln in Soße	PK	1.99	9

Beispiel matse\_mhist

## Inline Views

---

### Intention

- Außer Subselects im WHERE oder HAVING Teil zu verwenden, ist es ebenso möglich, sogenannte Inline Views bei FROM oder JOIN zu benutzen.

### Beispiel

- Hier werden alle Warengruppen inkl. eines aus den Produkten ermittelten Durchschnittspreises ausgegeben.
- Die im Subselect ermittelte Ergebnismenge wird unter dem Alias Q angesprochen und dort der Durchschnittspreis mit A. Q verhält sich wie eine eigene Tabelle bzw. eine eigene sog. View (folgt später).

```
SELECT W.* , Q.A FROM warengruppe W, (
    SELECT P.warengruppe_id, avg(P.stueckpreis) A
    FROM produkt P GROUP BY warengruppe_id
) Q
WHERE W.id=Q.warengruppe_id;
```

	■ id ■ bezeichnung	■ A ■
1	1 TK	2.390000
2	2 Obst, Gemüse	0.737500
3	3 Wurst, Aufschnitt	2.170000
4	4 Milchprodukte	0.920000
5	5 Kaltgetränke	1.447500
6	6 Grillgut	3.276667

Beispiel matse\_mhist

## Correlated Subselects

### Intention

- In Subselects auf die äußere Abfrage verweisen, sog. *Correlated Subselects*.

### Beispiel

- Hier werden alle Produkte ermittelt, deren Preis über dem Durchschnitt der entsprechenden Warengruppe liegt.
- Man beachte die Referenz auf die äußere Abfrage `P.warengruppe_id` im Subselect.

```
SELECT bezeichnung, stueckpreis, warengruppe_id
  FROM produkt P WHERE P.stueckpreis > (
    SELECT avg(stueckpreis) FROM produkt
      WHERE warengruppe_id=P.warengruppe_id
  );
```

	bezeichnung	stueckpreis	warengruppe_id
1	Nudelpfanne	3.29	1
2	Bananen	1.29	2
3	Knoblauch	0.98	2
4	Frikadellen	2.35	3
5	Quark	0.99	4
6	Joghurt	0.98	4

Beispiel matse\_mhist

## Subselects und EXISTS

---

### Intention

- Daten selektieren, die von der Existenz anderer Daten abhängig sind.

### Beispiel

- Es werden alle Mitarbeiter gesucht, die auch Vorgesetzter eines Mitarbeiters sind.
- Im Prinzip würde ein  
    SELECT 'X' FROM ...  
im Subselect ausreichen.
- Analog ist NOT EXISTS möglich.

```
SELECT M.id, M.name FROM mitarbeiter M
WHERE EXISTS (
    SELECT id FROM mitarbeiter
    WHERE vorgesetzter_mitarbeiter_id=M.id
);
```

	id	name
1	1	Mia
2	2	Ben
3	12	Leon
4	14	Luis
5	16	Lukas
6	20	Leonie

Beispiel matse\_mhist

## CHECK

---

### Aufgaben

Die benötigten Tabellen entstammen dem `matse_mhist` Schema.

1. Welche Produkte haben den gleichen Preis wie 'Chips'?

In der Ausgabe sollen 'Chips' nicht vorkommen.

bezeichnung	stueckpreis
Spinat	1.99
Fischstäbchen	1.99
Fleischwurst	1.99
Nudeln in Soße	1.99

2. Welche Warengruppen haben genau so viele Produkte wie die Warengruppe 2?

Geben Sie die Warengruppe, deren Namen und die jeweilige Anzahl Produkte aus.

warengruppe_id	bezeichnung	anzahl
2	Möhren	4
5	Cola light	4
9	Nudeln in Soße	4
10	Reisfladen	4

3. Welche Produkte haben innerhalb einer Warengruppe einen gleichen Preis?

warengruppe_id	bezeichnung	stueckpreis
1	Spinat	1.99
1	Fischstäbchen	1.99
5	Nerd Bull	1.50
5	Cola light	1.50

## CHECK

---

### Aufgaben

4. Sehen Sie sich zuerst die Relationen abteilung, rolle und arbeitet\_in\_als an.

Welcher Mitarbeiter arbeitet als 'Auszubildender' im Marketing?

5. Welche Mitarbeiter haben nur eine Gesamtstundenzahl von 5?

Geben Sie Name und Jahresgehalt aus.

6. Sehen Sie sich zuerst die Relationen bestellung und besteht\_aus an.

Welche Mitarbeiter haben schon eine Bestellung vermittelt?

id	name
23	Tim

mitarbeiter_id	name	jahresgehalt	sum
5	Hannah	5000.00	5.00
6	Luka	5000.00	5.00

name	id
Anna	1
Anna	3
Finn	2

## CHECK

---

### Aufgaben

7. Welche Produkte wurden noch nicht bestellt?
8. Sammeln Sie aus `besteht_aus` und `produkt` die Gesamtpreise je Bestellung und Produkt und summieren Sie danach die Bestellungen zum jeweiligen Gesamtpreis.
9. Gehen Sie jetzt von der Relation `bestellung` aus und verwenden Sie die vorherige Summentabelle analog der Inline Views als Inner Join zusammen mit der Relation `kunde` und erzeugen Sie eine Gesamtübersicht aller Bestellungen in der nebenstehenden Form.

<code>id</code>	<code>bezeichnung</code>	<code>warengruppe_id</code>	<code>stueckpreis</code>	<code>einheit</code>
1	Spinat	1	1.99	PK
2	Vier Käse Pizza	1	2.39	ST
3	Spinatpizza	1	2.29	ST
4	Fischstäbchen	1	1.99	PK
5	Nudelpfanne	1	3.29	PK
6	Möhren	2	0.39	KG

<code>bestellung_id</code>	<code>sum</code>
1	50.7600
2	83.8000
3	42.4800

<code>id</code>	<code>vom</code>	<code>name</code>	<code>sum</code>
1	2014-09-12 00:00:00	Sparkasse	50.7600
2	2014-08-30 00:00:00	Sparkasse	83.8000
3	2014-09-11 00:00:00	E.ON	42.4800

# UNIT 0X05

## SCHEMAS UND TABELLEN

## Schemas und Tabellen

---

### Lernziel

- Schemas und Tabellen im DBMS anlegen, verändern, löschen.
- Vorschau: Daten in Tabellen anlegen.

## Schemas

---

### Intention

- Vorhandene Schemas bzw. Datenbanken abfragen.

### Anmerkung

- Neben den eigenen Schemas gibt es immer auch DBMS-spezifische Schemas wie `information_schema`, in denen das DBMS eigene Daten hält, etwa die verschiedenen Speichermodelle.
- Die Verwaltung der Schemas und der internen Strukturen ist DBMS-spezifisch.

```
SHOW SCHEMAS;  
SHOW DATABASES;
```

```
Database  
information_schema  
kemper  
matse_algebra  
matse_mhist  
mysql  
oracle_hr  
performance_schema
```

## Schemas verwalten

### Intention

- Schema anlegen, Default-Schema festlegen und wieder löschen.
- Achtung: Ggf. muss in DBMS-Tools, etwa DataGrip, der Datenbestand aktualisiert werden (Refresh) und/oder die neuen Schemas eingeblendet werden!

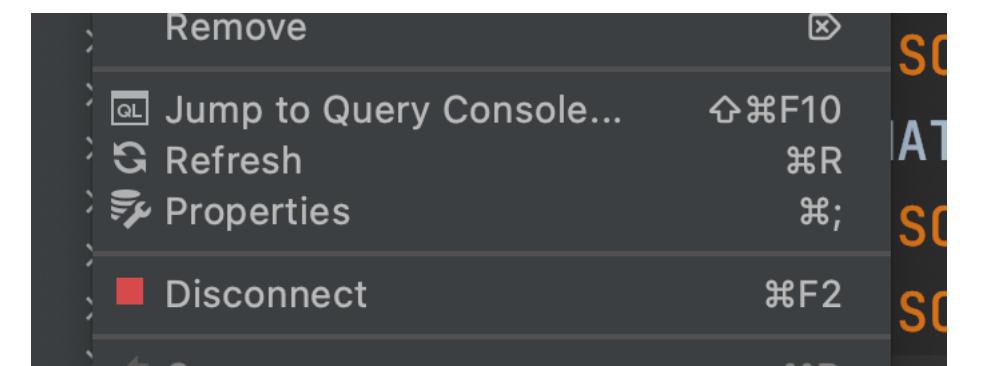
### Anmerkung

- Achtung: Löschen eines Schemas löscht alle Tabellen und Daten!

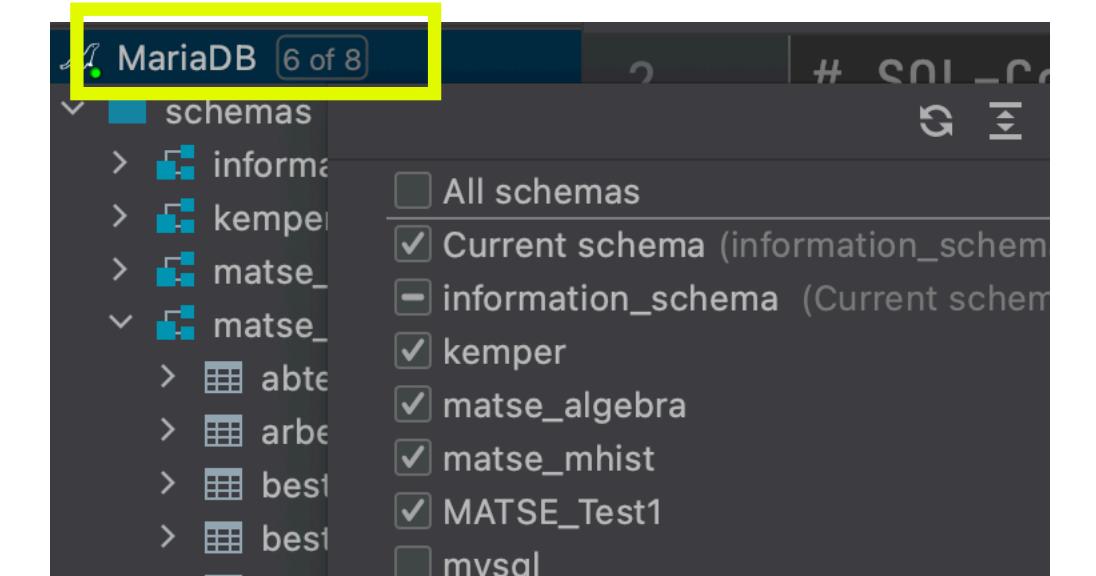
### Beispiel

```
CREATE SCHEMA MATSE_Test1;
SHOW SCHEMAS LIKE '%matse%';
USE MATSE_Test1;
DROP SCHEMA MATSE_Test1;
SHOW SCHEMAS;
```

```
Database (%matse%)
MATSE_Test1
matse_algebra
matse_mhist
```



Kontextmenü Datenbank



Angezeigte Schemas  
in DataGrip

## Tabellen

---

### Intention

- Vorhandene Tabellen in einem Schema abfragen.

### Beispiel

- Der zweite Befehl gibt nur die Tabellen aus, die mit 's' anfangen.
- In diesem DBMS spielt Groß- und Klein-schreibung keine Rolle.

```
SHOW TABLES FROM matse_mhist;  
SHOW TABLES FROM matse_mhist like 's%';
```

Tables_in_matse_mhist
abteilung
arbeitet_in_als
besteht_aus
bestellung
kennt
kunde

Tables_in_matse_mhist (s%)
sitzt_am
standort

Beispiel matse\_mhist

## Tabellenstruktur, Attribute/Spalten

---

### Intention

- Vorhandene Tabellenstruktur, d.h. u.a. Attribute/Spalten einer Tabelle abfragen.

### Beispiel

- Alle Befehle liefern dasselbe Resultat.
- Der erste (auskommentierte) Befehl wird als Fehler im Editor angezeigt, funktioniert aber...
- Angezeigt werden die Attribute mit Eigenschaften, also Namen, Datentyp und weiteren Informationen. Das wird nochmal beim Anlegen erläutert.

```
# SHOW COLUMNS FROM abteilung FROM matse_mhist;
SHOW COLUMNS FROM matse_mhist.abteilung;
DESCRIBE matse_mhist.abteilung;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	<null>	auto_increment
name	varchar(100)	YES		<null>	

Beispiel matse\_mhist

## Tabellen anlegen

---

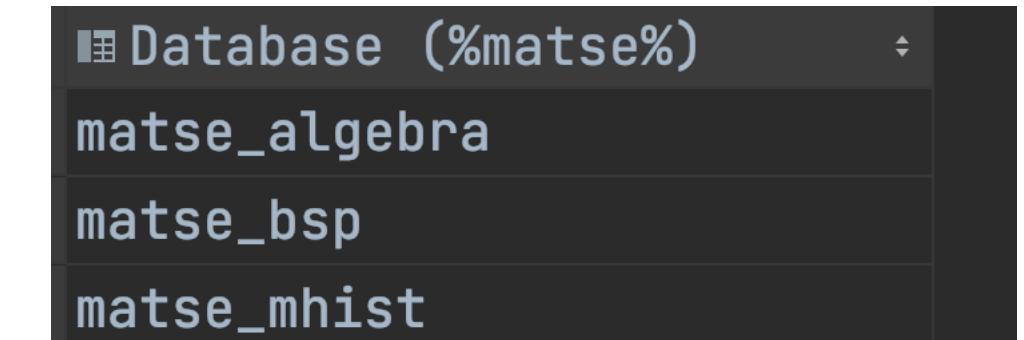
### Intention

- Tabellen erzeugen.
- Achtung: Vor strukturellen Änderungen an eigenen Schemas und Tabellen ein Backup erstellen (wie bei der Erstinstallation gezeigt) - besser ist das.

### Vorbereitung

- Bei den folgenden Beispielen in einem (temporären) Schema, etwa matse\_bsp, arbeiten, welches nachher wieder komplett gelöscht werden kann.
- Änderungen besser nicht in matse\_mhist durchführen!
- Erinnerung: Nach programmatischen Änderungen an den Tabellenstrukturen ggf. aktualisieren (Refresh).

```
CREATE SCHEMA matse_bsp;
SHOW SCHEMAS LIKE '%matse%';
USE matse_bsp;
```



The screenshot shows a dark-themed MySQL Workbench interface. A dropdown menu is open under the 'Database' heading, displaying several schema names: matse\_algebra, matse\_bsp, and matse\_mhist. The 'matse\_bsp' schema is highlighted, indicating it is the current active database.

Beispiel matse\_bsp

## Tabellen anlegen

---

### Beispiel

- Angegeben sind beim create table die einzelnen Spalten bzw. Attribute mit ihren jeweiligen Datentypen und Eigenschaften wie not null oder Defaultwerten.
- Bzgl. der Datentypen int, varchar etc. auf den Vorlesungsfolien oder in der DBMS-Dokumentation nachsehen.
- primary key legt den Schlüssel fest.
- unique not null muss eindeutig und darf nicht NULL sein.
- default legt Defaultwerte fest, now erzeugt einen aktuellen Zeitstempel (DBMS-spezifische Funktion).
- boolean ist tinyint(1), true ist 1.

```
CREATE TABLE objekte (
    id int primary key,
    name char(10) unique not null,
    kommentar varchar(255),
    nummer int(5),
    zahl decimal(8,3) default 0.0,
    erzeugt datetime default now(),
    wichtig boolean not null default true
);
SHOW COLUMNS FROM matse_bsp.objekte;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	<null>	
name	char(10)	NO	UNI	<null>	
kommentar	varchar(255)	YES		<null>	
nummer	int(5)	YES		<null>	
zahl	decimal(8,3)	YES		0.000	
erzeugt	datetime	YES		current_timestamp()	
wichtig	tinyint(1)	NO		1	

Beispiel matse\_bsp

## Tabellen anlegen

### Beispiel Fortsetzung

- Um zu sehen, was die Eigenschaften der Attribute auslösen, werden Datensätze angelegt. Das Kommando `insert` folgt noch, aber die Idee sollte zu sehen sein.
- Die `id` muss explizit angegeben werden, ebenso Attribute, die nicht `NULL` sein dürfen.
- Attribute, die beim `insert` keinen Wert zugewiesen bekommen, erhalten entweder ihren Defaultwert oder werden zu `NULL`.

```
INSERT INTO objekte (id, name) VALUES ('1', 'mueller');
INSERT INTO objekte (id, name, nummer) VALUES ('2', 'meier', '3');
SELECT * FROM objekte;
```

#	id	name	kommentar	nummer	zahl	erzeugt	wichtig
1	mueller	<null>	<null>	<null>	0.000	2021-01-01 11:50:12	1
2	meier	<null>		3	0.000	2021-01-01 11:50:13	1

Beispiel matse\_bsp

## Tabellen modifizieren

---

### Intention

- Tabellen ändern und Spalten bzw. Attribute hinzufügen, ändern oder löschen.

### Beispiele

- Die Spalten `image` und `eps` werden hinzugefügt und `eps` auch mit einem Defaultwert versehen.
- Der Datentyp `blob` kann ein großes Datenobjekt aufnehmen, z.B. eine Musikdatei (binary large object).

```
ALTER TABLE objekte ADD (
    image blob,
    eps double default 0.01
);
SELECT * FROM objekte;
SHOW COLUMNS FROM matse_bsp.objekte;
```

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>int(11)</code>	<code>NO</code>	<code>PRI</code>	<code>&lt;null&gt;</code>	
<code>name</code>	<code>char(10)</code>	<code>NO</code>	<code>UNI</code>	<code>&lt;null&gt;</code>	
<code>kommentar</code>	<code>varchar(255)</code>	<code>YES</code>		<code>&lt;null&gt;</code>	
<code>nummer</code>	<code>int(5)</code>	<code>YES</code>		<code>&lt;null&gt;</code>	
<code>zahl</code>	<code>decimal(8,3)</code>	<code>YES</code>		<code>0.000</code>	
<code>erzeugt</code>	<code>datetime</code>	<code>YES</code>		<code>current_timestamp()</code>	
<code>wichtig</code>	<code>tinyint(1)</code>	<code>NO</code>		<code>1</code>	
<code>image</code>	<code>blob</code>	<code>YES</code>		<code>&lt;null&gt;</code>	
<code>eps</code>	<code>double</code>	<code>YES</code>		<code>0.01</code>	

Beispiel matse\_bsp

# Tabellen modifizieren

---

## Beispiele

- Der erste Befehl dient der Kompatibilität mit Oracle und ändert den Datentyp und den Defaultwert.
- Der zweite Befehl ändert zusätzlich den Namen des Attributs.
- Da schon Werte in eps bzw. feps enthalten sind, wird zwar der Defaultwert geändert, nicht jedoch der Wert der vorhandenen Entities.

```
# Defaultwert ändern
ALTER TABLE objekte MODIFY eps float default 0.002;
SELECT * FROM objekte;
SHOW COLUMNS FROM matse_bsp.objekte;
# Name und Defaultwert ändern
ALTER TABLE objekte CHANGE eps feps float default 0.003;
SELECT * FROM objekte;
SHOW COLUMNS FROM matse_bsp.objekte;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	<null>	
name	char(10)	NO	UNI	<null>	
kommentar	varchar(255)	YES		<null>	
nummer	int(5)	YES		<null>	
zahl	decimal(8,3)	YES		0.000	
erzeugt	datetime	YES		current_timestamp()	
wichtig	tinyint(1)	NO		1	
image	blob	YES		<null>	
feps	float	YES		0.003	

Beispiel matse\_bsp

## Tabellen modifizieren

---

### Beispiele

- Die Attribute `feps` und `image` werden gelöscht. Die übrigen Werte bleiben davon unberührt.
- In manchen DBMS können Attribute als `unused` gekennzeichnet werden.

```
ALTER TABLE objekte DROP feps;
ALTER TABLE objekte DROP image;
SELECT * FROM objekte;
SHOW COLUMNS FROM matse_bsp.objekte;
```

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>int(11)</code>	<code>NO</code>	<code>PRI</code>	<code>&lt;null&gt;</code>	
<code>name</code>	<code>char(10)</code>	<code>NO</code>	<code>UNI</code>	<code>&lt;null&gt;</code>	
<code>kommentar</code>	<code>varchar(255)</code>	<code>YES</code>		<code>&lt;null&gt;</code>	
<code>nummer</code>	<code>int(5)</code>	<code>YES</code>		<code>&lt;null&gt;</code>	
<code>zahl</code>	<code>decimal(8,3)</code>	<code>YES</code>		<code>0.000</code>	
<code>erzeugt</code>	<code>datetime</code>	<code>YES</code>		<code>current_timestamp()</code>	
<code>wichtig</code>	<code>tinyint(1)</code>	<code>NO</code>		<code>1</code>	

Beispiel matse\_bsp

## Tabellen löschen

---

### Intention

- Tabellen umbenennen, leeren und löschen.

### Beispiel

- Der erste Befehl nennt Tabelle `objekte` in `elemente` um.
- Der Befehl `TRUNCATE` löscht alle Datensätze, lässt aber die Tabelle selber bestehen.
- Achtung: Löschen einer Tabelle mit `DROP` löscht neben der Tabelle auch alle Daten!

```
ALTER TABLE objekte RENAME TO elemente;
SELECT * FROM elemente;

TRUNCATE TABLE elemente;
SELECT * FROM elemente;

DROP TABLE elemente;
SHOW TABLES FROM matse_bsp;
```

Beispiel matse\_bsp

## Tabellen löschen

### Intention

- Tabellen mit Fremdschlüsseln, d.h. mit Attributen, die Datensätze in anderen Tabellen referenzieren, anlegen → **Constraints**.

### Beispiel

- Das Beispiel modelliert Personen und ihre Tiere, wobei eine Person für mehrere Tiere verantwortlich sein kann.
- Die Tabelle person wird nachfolgend durch die Tabelle tier referenziert.
- Beim Anlegen wird der PRIMARY KEY hier nicht beim Attribut, sondern am Ende hinzugefügt.

```
CREATE TABLE person (
    person_id INT NOT NULL,
    name VARCHAR(50) NOT NULL,
    PRIMARY KEY (person_id)
);
INSERT INTO person (person_id, name) VALUES ('11', 'MIA');
INSERT INTO person (person_id, name) VALUES ('12', 'LEA');
SELECT * FROM person;
```

person_id	name
11	MIA
12	LEA

Beispiel matse\_bsp

## Tabellen löschen

### Beispiel Fortsetzung

- `person_id` ist der Fremdschlüssel, der auf das gleichnamige Attribut in der Tabelle `person` verweist. Hier darf der Wert auch `NULL` sein, falls es (noch) keine Person gibt (wie bei Bello).
- Der Constraint trägt den Namen `fk_person`.
- Bei manchen DBMS muss für jede Fremdschlüsselbeziehung auch ein sog. Index existieren, um z.B. Joins zu beschleunigen, hier `person_idx`.
- Beim Anlegen von Daten in `tier` kann optional ein Fremdschlüssel angegeben werden. Dieser kann `NULL` sein oder aber eine Person referenzieren.
- Das DBMS, genauer der Constraint, verhindert hierbei die Angabe eines nicht-existierenden Datensatzes.

```

CREATE TABLE tier (
tier_id INT NOT NULL,
name VARCHAR(50) NOT NULL,
person_id INT NULL,
PRIMARY KEY (tier_id),
INDEX person_idx (person_id ASC),
CONSTRAINT fk_person FOREIGN KEY
(person_id) REFERENCES person (person_id) );

INSERT INTO tier (tier_id, name, person_id) VALUES ('1', 'Wuff', '11');
INSERT INTO tier (tier_id, name) VALUES ('2', 'Bello');
SELECT * FROM tier M LEFT OUTER JOIN person F ON M.person_id=F.person_id;

```

tier_id	M.name	M.person_id	F.person_id	F.name
1	Wuff	11		MIA
2	Bello	<null>	<null>	<null>

Beispiel matse\_bsp

## CHECK

---

### Aufgaben

Die nachfolgenden Tabellen entstammen *nicht* dem matse\_mhist Schema.

1. Legen Sie ein *eigenes* Schema matse\_sport an und wählen Sie es als Default-Schema.
2. Erzeugen Sie in matse\_sport die im folgenden Diagramm angegebenen Tabellen via SQL-Befehle mit den korrekten Fremdschlüsselbeziehungen.

Starten Sie mit den einfachen Tabellen sportler, wettkampf und team und legen Sie danach die Relationen pfeift und nimmt\_teil\_an mit den Fremdschlüsselbeziehungen an. Die inhaltliche Erklärung folgt im Anschluss.

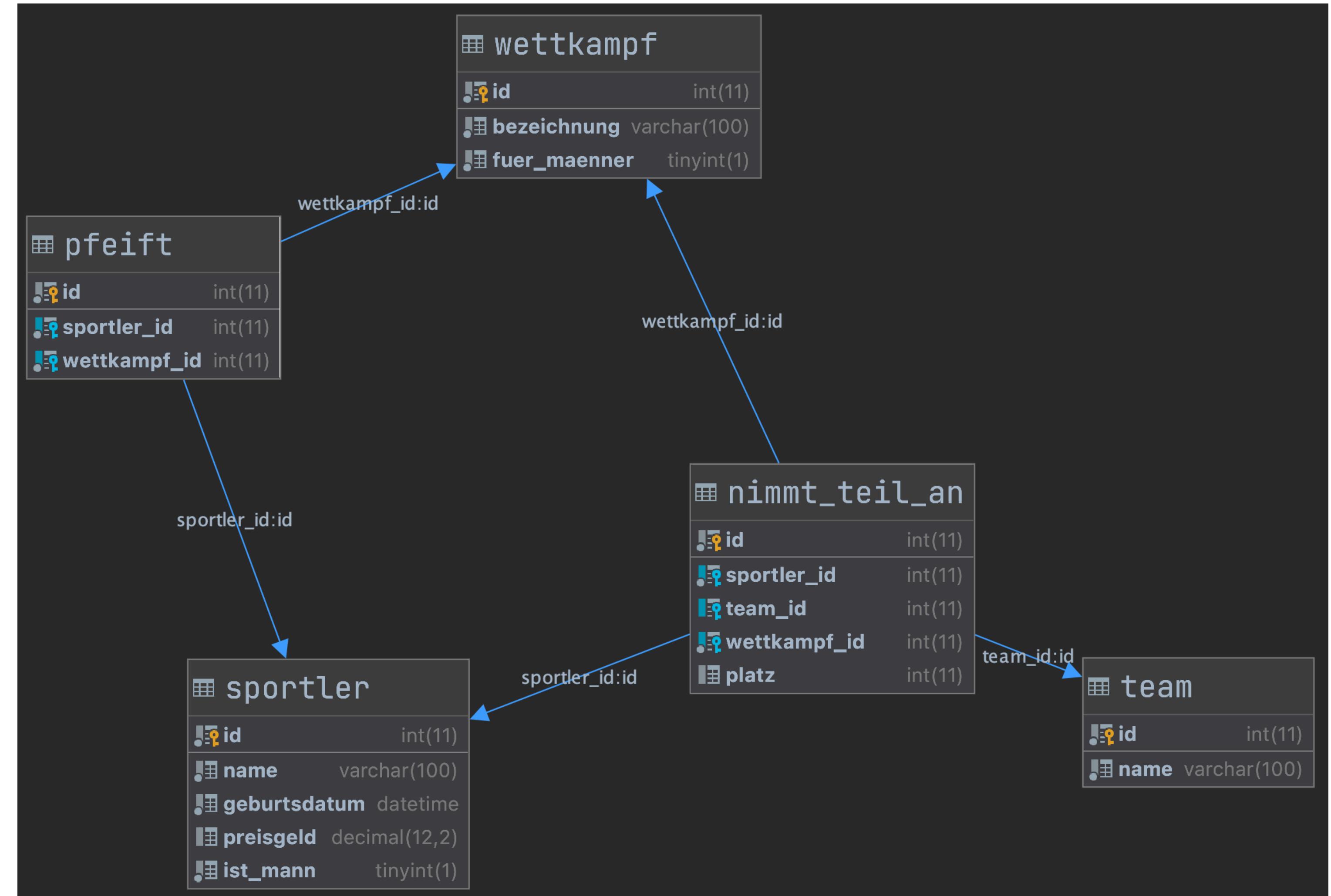
## CHECK

### Aufgaben

2. Fortsetzung. Es gelten nebenstehende Beziehungen in der Mini-Welt:



Diese Icons stehen für Primär- und Fremdschlüssel, welche in den Constraints berücksichtigt werden müssen.



## CHECK

---

### Aufgaben

2. Fortsetzung. Es gelten folgende Bedingungen in der Mini-Welt:

- ♦ Es gibt Sportler, Wettkämpfe und Teams mit den angegebenen Attributen. Darüber hinaus gibt es eine Relation, die beschreibt, wer in welchem Team an welchem Wettkampf teilnimmt und eine dafür, wer den jeweiligen Wettkampf pfeift.
- ♦ Alle Attribute der Sportler sind immer anzugeben. Per Default wird das Preisgeld auf 0 gesetzt und der Sportler ist ein Mann.
- ♦ Ein Wettkampf hat eine nicht-optionale Bezeichnung und ist für Männer oder Frauen; per Default ist er für Männer.
- ♦ Ein Team hat nur einen Namen.
- ♦ Aus Kostengründen gibt es keine eigenen Schiedsrichter, sondern es pfeift immer genau ein Sportler einen Wettkampf.

## CHECK

---

### Aufgaben

2. Fortsetzung. Es gelten folgende Bedingungen in der Mini-Welt:

- n Sportler können an m Wettkämpfen teilnehmen. Optional ist es ein Teamwettbewerb, dann ist zusätzlich das Team angegeben.

Erzeugen Sie folgende Datensätze in den jeweiligen Tabellen, z.B. per INSERT SQL-Befehl. Dieser hat, wie zuvor, immer so eine Form (z.B. für zwei Spalten):

```
INSERT INTO `tabelle` (`attribut1`, `attribut2`) VALUES ('wert1', 'wert2');
```

Es ist auch möglich, mehrere Daten mit einem Befehl einzufügen:

```
INSERT INTO `tabelle` VALUES ('attribut1','attribut2'),('attribut1', 'attribut2');
```

## CHECK

---

### Aufgaben

#### 2. Fortsetzung

id	bezeichnung	fuer_maenner
56	Tennis Vorrunde Doppel	0
98	Tennis Finale Einzel	1
99	Tennis Finale Einzel	0

Tabelle wettkampf

id	name	geburtsdatum	preisgeld	ist_mann
101	Anna	1990-02-01 00:00:00	0.00	0
102	Olga	1991-03-01 00:00:00	0.00	0
111	Enie	1992-04-01 00:00:00	100.00	0
112	Antje	1993-05-01 00:00:00	200.00	0
121	Boris	1990-06-01 00:00:00	3000.00	1
122	Ivan	1991-07-01 00:00:00	4000.00	1

Tabelle sportler

id	name
12345	Team NL
98765	Team PL

Tabelle team

id	sportler_id	wettkampf_id	team_id	platz
1	101	98765		56
2	102	98765		56
3	111	12345		56
4	112	12345		56
5	101	<null>		99
6	111	<null>		99
7	121	<null>		98
8	122	<null>		98

Tabelle nimmt\_teil\_an

id	sportler_id	wettkampf_id
1	121	56
2	122	99
3	101	98

Tabelle pfeift

## CHECK

---

### Aufgaben

Beantworten Sie folgende Fragen:

3. Welcher Sportler nimmt an mehr als einem Wettbewerb teil? Geben Sie jeweils Name und die jeweilige Anzahl an Wettbewerben dieses Sportlers aus.
4. Welche Sportler pfeifen keinen Wettbewerb? Geben Sie die Namen aus.
5. Welche Teams spielen in welchen Wettbewerben? Geben Sie die Teamnamen und die zugehörigen Wettbewerbsbezeichnungen aus.
6. Welche Sportler stehen in einem Finale? Geben Sie die Namen und die zugehörigen Finale (Wettbewerbsbezeichnungen) aus.

name	count(*)
Anna	2
Enie	2

name
Olga
Enie
Antje

name	bezeichnung
Team NL	Tennis Vorrunde Doppel
Team PL	Tennis Vorrunde Doppel

name	bezeichnung
Boris	Tennis Finale Einzel
Ivan	Tennis Finale Einzel
Anna	Tennis Finale Einzel
Enie	Tennis Finale Einzel

# UNIT 0x06

## ZEIT- UND DATUMSFUNKTIONEN

# Zeit- und Datumsfunktionen

---

## Lernziel

- Zeit- und Datumsfunktionen verwenden.

## Motivation

- Die Art, wie Zeit und Daten behandelt werden, ist ein steter Quell der Freude, weil es so viele Varianten, Datentypen und DBMS-spezifische Einstellungen gibt.

Wir wollen hier aber zumindest ein paar (MySQL/MariaDB-spezifische) Beispiele angeben, so dass eine Idee entstehen kann, wie diese Funktionen verwendet werden.

Für das eigene DBMS immer nachlesen.

Name	Description
<a href="#">ADDDATE()</a>	Add time values (intervals) to a date value
<a href="#">ADDTIME()</a>	Add time
<a href="#">CONVERT_TZ()</a>	Convert from one time zone to another
<a href="#">CURDATE()</a>	Return the current date
<a href="#">CURRENT_DATE()</a> , <a href="#">CURRENT_DATE</a>	Synonyms for CURDATE()
<a href="#">CURRENT_TIME()</a> , <a href="#">CURRENT_TIME</a>	Synonyms for CURTIME()
<a href="#">CURRENT_TIMESTAMP()</a> , <a href="#">CURRENT_TIMESTAMP</a>	Synonyms for NOW()
<a href="#">CURTIME()</a>	Return the current time
<a href="#">DATE()</a>	Extract the date part of a date or datetime expression
<a href="#">DATE_ADD()</a>	Add time values (intervals) to a date value
<a href="#">DATE_FORMAT()</a>	Format date as specified
<a href="#">DATE_SUB()</a>	Subtract a time value (interval) from a date
<a href="#">DATEDIFF()</a>	Subtract two dates
<a href="#">DAY()</a>	Synonym for DAYOFMONTH()
<a href="#">DAYNAME()</a>	Return the name of the weekday
<a href="#">DAYOFMONTH()</a>	Return the day of the month (0-31)
<a href="#">DAYOFWEEK()</a>	Return the weekday index of the argument
<a href="#">DAYOFYEAR()</a>	Return the day of the year (1-366)
<a href="#">EXTRACT()</a>	Extract part of a date
<a href="#">FROM_DAYS()</a>	Convert a day number to a date
<a href="#">FROM_UNIXTIME()</a>	Format Unix timestamp as a date
<a href="#">GET_FORMAT()</a>	Return a date format string
<a href="#">HOUR()</a>	Extract the hour

Ausschnitt Datumsfunktionen MySQL

## Zeitzonen

---

### Hintergrund

- Zeitangaben sind grundsätzlich abhängig von Zeitzonen, die entweder explizit angegeben werden oder vom System oder DBMS implizit verwendet werden. Wie diese aussehen, ist ganz schön hier <https://de.wikipedia.org/wiki/Zeitzone> zu sehen.
- Die lokalen Zeiten werden relativ zur koordinierten Weltzeit (englisch Coordinated Universal Time), kurz UTC, bezogen, wobei  $\text{UTC}\pm 0$  gerade die auf den Längengrad von Greenwich bezogene Zonenzeit ist. Die Unterschiede zwischen Universalzeit (UT) und UTC lassen wir hier aussen vor.
- Darüber hinaus ist noch Sommer- und Winterzeit zu berücksichtigen. So gilt im deutschsprachigen Raum im Winter die MEZ (UTC+1h) (Normalzeit), in den Sommermonaten jedoch die mitteleuropäische Sommerzeit (MESZ, UTC+2h).

## Interne Darstellung

### Hintergrund

- Wie ein Zeitpunkt intern dargestellt wird, ist noch einmal ein ganz eigenes Kapitel. Grundsätzlich könnte man meinen, dass das erst einmal keine Rolle spielt. Wenn jedoch z.B. sehr genaue, ältere oder ferne Zeitpunkte gespeichert werden sollen, so muss man sicherstellen, dass das interne Format das leisten kann.  
Als Beispiel diene die sog. Unixzeit.

Diese zählt die vergangenen Sekunden seit Donnerstag, dem 1. Januar 1970, 00:00 Uhr UTC (The Epoch).

Am 19. Januar 2038 um 3:14:08 Uhr UTC wird es daher bei Systemen, welche die Unixzeit in einer vorzeichenbehafteten 32-Bit-Variable speichern, zu einem Überlauf, und mithin zu einem Rücksprung kommen...

- YEAR: A one-byte integer
- DATE: A three-byte integer packed as  $YYYY \times 16 \times 32 + MM \times 32 + DD$
- TIME: A three-byte integer packed as  $DD \times 24 \times 3600 + HH \times 3600 + MM \times 60 + SS$
- TIMESTAMP: A four-byte integer representing seconds UTC since the epoch ('1970-01-01 00:00:00' UTC)**
- DATETIME: Eight bytes: A four-byte integer for date packed as  $YYYY \times 10000 + MM \times 100 + DD$  and a four-byte integer for time packed as  $HH \times 10000 + MM \times 100 + SS$

Ausschnitt Speicherformat MySQL <5.6.4.

## Interne Darstellung

---

### Anmerkungen

- Alle Zeiten, die von irgendeinem System geliefert werden, muss man unbedingt mit Blick auf die eigenen Anwendungsfälle untersuchen, etwa:
  - Weiß ich, was das jeweilige System/die verwendete Funktion liefert?  
(UTC vs. lokale Zeit inkl. Zeitumstellung)
  - Sind lokale Zeiten für mein Problem ausreichend?  
(heutiges Kinoprogramm vs. international koordinierter Raketenstart)
  - Welcher Zeitraum kommt vor?  
(heute +/- ein paar Jahre vs. Weihnachten zur Kreidezeit)
  - Welche Genauigkeit wird benötigt? Brauche ich nur Tage, oder auch Sekunden oder noch genauere Zeitstempel  
(Kinoprogramm vs. Messdaten).

## Zeitzonen

### Beispiele

- Vorab: Diese Einstellungen sind MySQL-spezifisch und was wir hier zeigen, ist nicht vollständig. Weiter hängt es auch noch an der Konfiguration des Systems, konkret daran, ob MySQL so konfiguriert ist, dass Zeitzonen als Name verfügbar sind...
- Es gibt eine lokale (session) und eine globale (global) Zeitzone, die entweder auf einen Wert wie 'Europe/Berlin' oder ein Offset eingestellt werden kann, z.B. '+02:00'.  
Wir betrachten jetzt nur die lokale Zeit. Den Effekt kann man im Beispiel beobachten, wobei `@@session.time_zone` die lokale Zeitzone und `now()` das aktuelle Datum inkl. Zeit ergibt.

```
SET SESSION time_zone = 'SYSTEM';
SELECT @@session.time_zone, now();

SET SESSION time_zone = 'Europe/Berlin';
SELECT @@session.time_zone, now();

SET time_zone = '+02:00';
SELECT @@session.time_zone, now();
```

@@session.time_zone	now()
SYSTEM	2021-01-07 11:52:02
Europe/Berlin	2021-01-07 12:52:19
+02:00	2021-01-07 13:52:31

## Zeit- und Datumsfunktionen

---

### Beispiele

- Aktuelles Datum und Zeit abfragen (Typ datetime) und in Datentyp date und time umwandeln.
- Ein Datum kann modifiziert werden, entweder über Funktionen wie date\_add oder per + oder -.
- Es kann auch eine Zeitspanne ermittelt werden, hier in Tagen bzw. Stunden.

```
SELECT now() 'N',
       cast(now() AS date) 'D',
       cast(now() AS time) 'T',
       cast(now() AS datetime) 'DT';
```

N	D	T	DT
2021-01-07 07:56:20	2021-01-07	07:56:20	2021-01-07 07:56:20

```
SELECT cast(now() AS date) + interval 2 year;
SELECT cast(date_add(now(), interval 2 year) as date);
SELECT cast(now() AS time) - interval 2 hour;
```

```
SELECT datediff(date_add(now(), interval 1 year),now());
SELECT timediff(NOW(), UTC_TIMESTAMP);
```

## Umwandlung

---

### Beispiele

- Der erste Ausdruck wandelt einen String, gegeben in Europäischem Format, in ein Datum um.
- Der zweite Ausdruck wandelt dagegen ein Datum in das Europäische Format um.
- Der Formatstring kann auch direkt angegeben werden, hier im letzten Beispiel mit Wochentag, Monatsname und 4-stelligem Jahr.

```
SELECT str_to_date('13.10.2014',get_format(date,'EUR'));
SELECT date_format('2014-10-13',get_format(date,'EUR'));
SELECT date_format('2014-10-13','%W %M %Y');
```

```
str_to_date('13.10.2014',get_format(date,'EUR'))
2014-10-13
```

```
date_format('2014-10-13',get_format(date,'EUR'))
13.10.2014
```

```
date_format('2014-10-13','%W %M %Y')
Monday October 2014
```

## CHECK

---

### Aufgaben

Verwenden Sie das Wettkampf-Schema der Aufgaben zuvor. Beantworten Sie dann folgende Fragen.

1. Welche Sportler sind Montagskinder (an einem Montag geboren)? Geben Sie jeweils den Namen und das Geburtsdatum aus.  
Hinweis: Durchstöbern Sie die Hilfeseite Ihres DBMS nach geeigneten Funktionen.
2. Welche Sportler sind jünger als 25? Geben Sie jeweils den Namen und das Alter aus.  
Hinweis: Sie brauchen Schaltjahre nicht zu berücksichtigen.
3. Formulieren Sie Aufgabe 2 so, dass die Berechnung des Alters nur einmal in dem SQL-Befehl auftaucht.
4. Welcher Wettkampf ist der mit dem höchsten Durchschnittsalter seiner Teilnehmer?  
Geben Sie die Bezeichnung dieses Wettkampfes mit dem Durchschnittsalter aus.

UNIT 0x07

VIEWS

## Views

### **Lernziel**

- Views (Sichten) anlegen und löschen.

## Views

### Intention

- Views (Sichten) anlegen und löschen.

```
SELECT P.bezeichnung, P.stueckpreis AS 'preis'  
FROM produkt P where P.bezeichnung like '%pizza%';
```

Abfrage der Pizzen

### Beispiel

- Die Abfrage filtert alle Pizzen aus dem Sortiment. Das wollen wir als view mit Namen `pizzen` formulieren.
- Falls es `pizzen` schon gibt, wird es neu definiert und im Anschluss wie eine Tabelle verwendet. `pizzen` findet sich dann als view auch im Kontextmenü.
- Um `pizzen` zu löschen, nutzt man `drop view`.

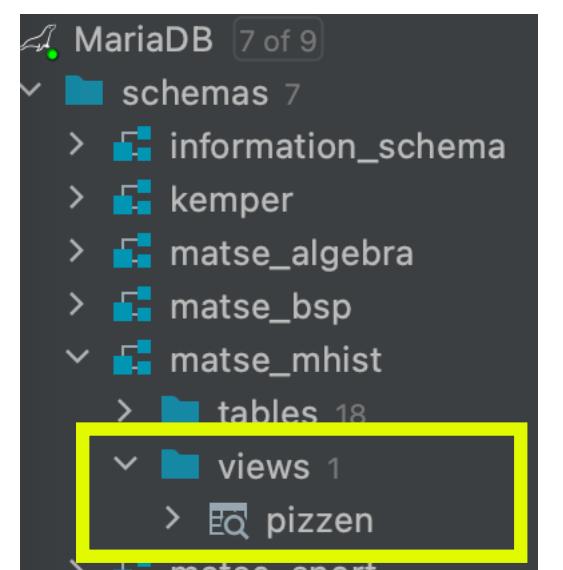
```
CREATE OR REPLACE VIEW pizzen AS (  
    SELECT P.bezeichnung, P.stueckpreis AS 'preis'  
    FROM produkt P where P.bezeichnung like '%pizza%'  
);
```

View `pizzen`

```
SELECT * FROM pizzen;
```

Verwendung wie Tabelle

bezeichnung	preis
Vier Käse Pizza	2.39
Spinatpizza	2.29



```
DROP VIEW pizzen;
```

## CHECK

---

### Aufgaben

Hier wird das `matse_mhist` Schema verwendet.

1. Wir betrachten *Produkte* und nähern uns einem funktionierenden `select`-Befehl, den wir dann abschliessend in ein View packen.
  - (a) Warm-Up: Suchen Sie alle Produkte der Warengruppe 1 und geben Sie hier Bezeichnung und Stückpreis als Preis aus.
  - (b) Erweitern Sie (a) und suchen nicht direkt nach Produkten der Warengruppe 1, sondern nach Produkten der Warengruppe mit dem Namen 'TK' (hat natürlich die `id` 1). Nutzen Sie hierfür ein Subselect.
  - (c) Formulieren Sie (b) so um, dass die Warengruppe 'TK' mittels `with` gefunden wird.
  - (d) Erzeugen Sie ein View `only_tk` mit einer der drei Abfragen (a)-(c). Nutzen Sie es in einem `select` und löschen Sie es am Ende wieder.

bezeichnung	preis
Spinat	1.99
Vier Käse Pizza	2.39
Spinatpizza	2.29
Fischstäbchen	1.99
Nudelpfanne	3.29

## CHECK

---

### Aufgaben

Hier wird das matse\_mhist Schema verwendet.

2. Wir betrachten *Bestellungen* und nähern uns einem funktionierenden select-Befehl, den wir dann abschliessend in ein View packen.
- (a) Sehen Sie sich die Tabelle bestellung an, danach besteht \_ aus und verstehen Sie, wie sich eine Bestellung als Menge einzelner Produkte zusammensetzt.

#id	kunde_id	vom
1	30	2014-09-12 00:00:00
2	30	2014-08-30 00:00:00
3	32	2014-09-11 00:00:00

#id	bestellung_id	produkt_id	einheiten
1	1	15	12.00
2	1	16	6.00
3	1	18	24.00

- (b) Zusammen mit dem Produktpreis kann man so den Gesamtpreis jeder Bestellung ausrechnen. Formulieren Sie dafür ein entsprechendes select mit diesem Ergebnis und ergänzen Sie es (join) zu

bestellung_id	summe
1	50.7600
2	83.8000
3	42.4800

#id	vom	name	summe
1	2014-09-12	Sparkasse	50.7600
2	2014-08-30	Sparkasse	83.8000
3	2014-09-11	E.ON	42.4800

## CHECK

---

### Aufgaben

2. Fortsetzung.

- (c) Formulieren Sie das um Kundenname und Bestelldatum ergänzte select aus (b) als View, nutzen Sie es und löschen Sie es wieder.
- (d) Je nach DBMS und/oder Version kann es zu Problemen kommen, wenn das View wiederum ein eingebettetes View, etwa über ein Subselect, enthält.  
Für diesen Fehlerfall, und/oder wenn Ihr View aus (c) so ein Subselect, enthält, formulieren Sie Ihr View so um, dass mehrere geschachtelte Views verwendet werden.

# UNIT 0x08

# TRANSAKTIONEN

## Transaktionen

---

### Lernziel

- Transactions verwenden.
- Datenbankoperationen nur als Ganzes anwenden oder verwerfen, vgl. Kontobewegung bei einer Bank.
- Stichwort ACID bzw. Atomarität.

# Transaktionen

## Vorbereitung

- Um zu verstehen, welche Effekte die mit einer Transaktion in Verbindung stehenden Kommandos commit und rollback haben, benötigen wir eine einfache Ausgangssituation, die wir leicht wiederherstellen können.
- Dazu nutzen wir das Schema matse\_bsp und die beiden Tabellen person und tier aus 0x05. Im Skript db\_ex\_08 sind die erzeugenden Befehle auch angegeben. Die Daten sehen dann so aus:

Database (%matse%)
matse_algebra
<b>matse_bsp</b>
matse_mhist
matse_sport

Schema matse\_bsp

Tables_in_matse_bsp
person
tier

Tabellen matse\_bsp

person_id	name
11	MIA
12	LEA

Tabelle person

tier_id	name	person_id
1	Wuff	11
2	Bello	<null>

Tabelle tier

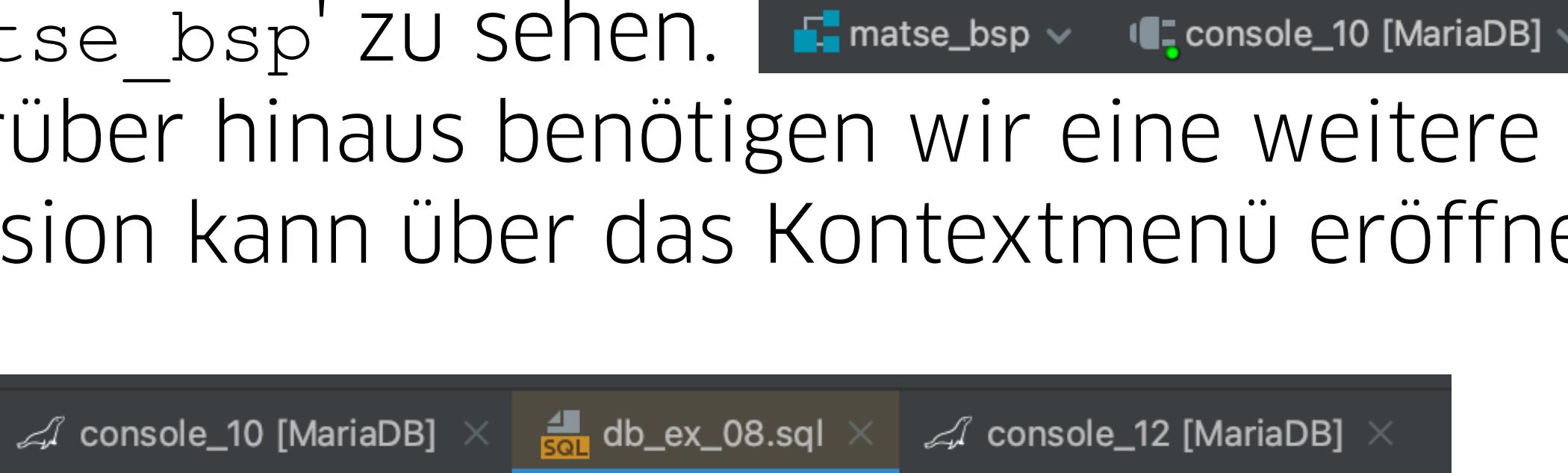
tier_id	M.name	M.person_id	F.person_id	F.name
1	Wuff	11	11	MIA
2	Bello	<null>	<null>	<null>

Outer Join tier person

## Transaktionen

---

### Vorbereitung

- Weiterhin gibt es eine Einstellung bei MySQL/MariaDB, die dazu führt, dass jeder Befehl sofort verarbeitet wird. Diese Einstellung `SET autocommit=1;` ist normalerweise aktiv, aber um sicher zu gehen, setzen wir sie im Skript explizit erstmal auf 1.
- Speziell in DataGrip ist es so, dass geladene Skripte in einer bestimmten Session ausgeführt werden. Die wird beim Laden des Skripts manuell zugewiesen oder automatisch zugeordnet und ist links am Bildschirm neben dem aktiven Schema 'matse\_ bsp' zu sehen.  In diesem Beispiel ist es 'console 10'. Darüber hinaus benötigen wir eine weitere Console, hier 'console 12'. Diese zweite Session kann über das Kontextmenü eröffnet werden: 'Jump to Query Console' oder ähnlich.



Editorfenster: zwei Sessions und das Skript

```
use matse_bsp;  
select * from person;  
select * from tier;
```

SQL in beiden Sessions,  
um die jeweilige aktuelle  
Datensicht zu sehen.

## Transaktionen

---

### Intention

- Wir legen jetzt Daten in den Tabellen an und sehen in den verschiedenen Sessions, was dort jeweils zu sehen ist.
- Dann können wir die Änderungen übernehmen (commit) oder den Ursprungszustand wieder herstellen (rollback).

# Transaktionen

---

## Beispiel 1

- Wir gehen von unseren Ausgangsdaten aus.
- Step 1: `START TRANSACTION;  
INSERT INTO person (person_id, name) VALUES (21, 'Max');`

Daten danach:

person_id	name
11	MIA
12	LEA
21	Max

console\_10

person_id	name
11	MIA
12	LEA

console\_12

Man sieht, dass in der ersten Session, in der das Skript ausgeführt wird, die Änderung direkt sichtbar ist, in der anderen nicht. Das ist auch die Sicht eines zweien Benutzers.

- Step 2: `ROLLBACK;`

Daten danach:

person_id	name
12	LEA

console\_10

person_id	name
11	MIA
12	LEA

console\_12

Hier führt ROLLBACK dazu, dass alle Änderungen wieder zurückgenommen wurden.

# Transaktionen

---

## Beispiel 2

- Wir gehen wieder von unseren Ausgangsdaten aus.
- Step 1: `START TRANSACTION;  
INSERT INTO person (person_id, name) VALUES (21, 'Max');`

Daten danach:

person_id	name
11	MIA
12	LEA
21	Max

console\_10

person_id	name
11	MIA
12	LEA

console\_12

Man sieht wie zuvor, dass in der ersten Session, in der das Skript ausgeführt wird, die Änderung direkt sichtbar ist, in der anderen nicht.

- Step 2: `COMMIT;`

Daten danach:

person_id	name
11	MIA
12	LEA
21	Max

console\_10

person_id	name
11	MIA
12	LEA
21	Max

console\_12

Hier führt jetzt COMMIT dazu, dass alle Änderungen in allen Sessions, und so auch für alle Benutzer, zu sehen sind. Achtung, den neuen Datensatz wieder löschen.

## Transaktionen

---

### Beispiel 3

- Wir gehen wieder von unseren Ausgangsdaten aus.
- Step 1: Hier verwenden wir eine falsche person\_id=22, was direkt zu einem Fehler führt, da der Fremdschlüssel fehlt.

```
START TRANSACTION;  
INSERT INTO person (person_id, name) VALUES (21, 'Max');  
INSERT INTO tier (tier_id, name, person_id) VALUES (5, 'Mini', 22);  
ROLLBACK;
```

[23000][1452] (conn=32) Cannot add or update a child row: a foreign key constraint fails ('matse\_bsp`.`tier`,  
CONSTRAINT `fk\_person` FOREIGN KEY (`person\_id`) REFERENCES `person` (`person\_id`))

- Step 2: **ROLLBACK;**  
Durch die Verwendung einer Transaktion und die Wiederherstellung im Fehlerfall ist zwar der Fehler nicht behoben, aber für alle anderen ist die Datenbank immer noch in dem ursprünglichen Zustand. Man hat also keine halb-eingefügten Daten, sondern ganz-oder-gar-nicht - das war ja genau das Ziel.

# Transaktionen

## Beispiel 4

- In Beispiel 3 wurde bewusst ein nicht existierender Fremdschlüssel verwendet, aber diese Fehlersituation tritt z.B. auch auf, wenn Tabellen aus externen Quellen in 'falscher' Reihenfolge in das DBMS eingelesen werden. Um das Problem zu lösen, kann man versuchen, eine logisch passende Reihenfolge zu finden, so dass die Daten sukzessive aufgebaut werden und alle Referenzen beim Einspielen gültig sind. Gibt es in den Daten aber wechselseitige Referenzen zwischen den Tabellen, ist das so nicht möglich. Dann kann man für einen 'kurzen' Moment die Prüfung auf referentielle Integrität ausschalten, die Daten einspielen und so dieses Problem umgehen.

Das Beispiel 4 im SQL-Skript stellt dies dar und funktioniert wie die bisherigen.

- Vergisst man das Zurückstellen des `foreign_key_checks` oder ist unvorsichtig, dann kann man unbekannte Fremdschlüssel einfügen, ohne dass das zunächst auffällt...

```
SET foreign_key_checks = 0;
START TRANSACTION;
INSERT INTO tier (tier_id, name, person_id) VALUES (5, 'Mini', 21);
INSERT INTO person (person_id, name) VALUES (21, 'Max');
COMMIT;
# unbedingt wieder herstellen
SET foreign_key_checks = 1;
```

## Transaktionen

---

### Beispiel 5

- Die in der Vorbereitung schon erwähnte Einstellung `autocommit=1` führt kurz gesagt dazu, jeder abgesetzte Befehl sofort mit einem impliziten `commit` versehen ist. Deswegen ist eine Datenänderung auch sofort in einer anderen Session zu sehen.
- Schaltet man die Einstellung aus `SET autocommit=0;` so muss erst ein `commit` erfolgen, damit die Änderungen in anderen Sessions zu sehen sind. Das System verhält sich so, als wenn ständig eine Transaktion aktiv ist, die erst bestätigt oder zurück gesetzt werden muss.
- Achtung: Dieser Effekt ist (auf meinem System) nur bei MySQL zu sehen. Bei MariaDB kann zwar die Einstellung `SET autocommit=0;` gesetzt werden, aber das System verhält sich dennoch so, als wenn `autocommit=1` gilt.

```
SET autocommit=0;
INSERT INTO person (person_id, name) VALUES (21, 'Max');
SELECT * FROM person;
ROLLBACK;
```

## Check

---

### Aufgaben

1. Nehmen Sie sich ein Schema und eine Tabelle Ihrer Wahl und starten Sie eine Transaktion. Fügen Sie nun Daten ein bzw. Ändern oder Löschen Sie Daten. Danach verwerfen Sie die Änderungen bzw. nehmen Sie sie an. Überzeugen Sie sich vom Ergebnis.

# UNIT 0x09

## DATEN ANLEGEN, VERÄNDERN, LÖSCHEN

## Daten anlegen, verändern, löschen

---

### Lernziel

- Datensätze anlegen/einfügen, verändern und löschen.

### Vorbereitung

- Wir nutzen das Wettkampf-Schema `matse_sport` aus den Aufgaben in Unit 0x05 und Unit 0x06. Falls Sie das Schema nicht (oder anders) angelegt haben, finden Sie die notwendigen Befehle auch in den Lösung zu Unit 0x05 (`db_ex_05.sql`).

## Daten anlegen, verändern, löschen

---

### Intention

- Datensätze einfügen/anlegen.

### Beispiel

- Hier werden nacheinander drei Datensätze angelegt. Argumente können entweder in Anführungsstrichen angegeben werden, oder, wenn es der Typ zulässt, auch ohne.
- Die Werte können auch Ergebnis eines Ausdrucks sein, hier z.B. das Geburtsdatum von Pete.

```
INSERT INTO sportler (id, name, geburtsdatum, ist_mann)
VALUES ('131', 'Pierre', '1991-12-24', '1');
INSERT INTO sportler (id, name, geburtsdatum, ist_mann)
VALUES (132, 'Rob', '1991-12-24', true);
INSERT INTO sportler (id, name, geburtsdatum)
VALUES (133, 'Pete', cast(now() AS date) - interval 20 year);
```

131	Pierre	1991-12-24 00:00:00	0.00	1
132	Rob	1991-12-24 00:00:00	0.00	1
133	Pete	2001-01-12 00:00:00	0.00	1

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Werte für Attribute aus anderen Datensätzen generieren.

### Beispiel

- Hier wird das Geburtsdatum für 'Marc' mittels eines Subselects aus dem Geburtsdatum von 'Pete' hergeleitet (Zwillinge über Mitternacht).
- Das Ergebnis des Subselects muss aus einem Wert bestehen.

```
SELECT S.name, S.geburtsdatum+interval 1 day  
FROM sportler S WHERE S.name='Pete';
```

Pete	2001-01-13 00:00:00
------	---------------------

```
INSERT INTO sportler (id, name, geburtsdatum)  
VALUES (134, 'Marc', (  
    SELECT S.geburtsdatum+interval 1 day FROM sportler S  
    WHERE S.name='Pete'  
));
```

133	Pete	2001-01-12 00:00:00	0.00	1
134	Marc	2001-01-13 00:00:00	0.00	1

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Mehrere Datensätze einfügen/anlegen.

### Beispiel

- Durch die Angabe mehrerer Wertetupel werden auch mehrere Datensätze angelegt.
- Es kann auch NULL angegeben werden, sofern es für das jeweilige Attribut erlaubt ist.

```
INSERT INTO sportler (id, name, geburtsdatum, preisgeld) VALUES
(135, 'Tick', '1992-02-21 15:00', 100.0),
(136, 'Trick', '1992-02-21 15:01', NULL),
(137, 'Track', '1992-02-21 15:02', 200.0);
```

135	Tick	1992-02-21 15:00:00	100.00	1
136	Trick	1992-02-21 15:01:00	<null>	1
137	Track	1992-02-21 15:02:00	200.00	1

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Ganze Datensätze aus anderen Datensätzen generieren.

### Beispiel

- Es ist auch möglich, ähnlich wie zuvor für einzelne Attribute, neue Datensätze aus einem Subselect zu generieren, wenn das Resultat aus mehreren Zeilen und Attributen besteht.
- In diesem Fall wird VALUES weggelassen.
- So kann man leicht Tabellen als 'Vorlage' für abgeleitete Daten verwenden.

```
SELECT S.id, S.name ,S.geburtsdatum  
FROM sportler S WHERE S.name like '%ck%';
```

ID	Name	Geburtsdatum
135	Tick	1992-02-21 15:00:00
136	Trick	1992-02-21 15:01:00
137	Track	1992-02-21 15:02:00

```
INSERT INTO sportler (id, name, geburtsdatum)  
SELECT S.id+3,concat(S.name,' Double'),S.geburtsdatum  
FROM sportler S WHERE S.name like '%ck';
```

138	Tick Double	1992-02-21 15:00:00	0.00	1
139	Trick Double	1992-02-21 15:01:00	0.00	1
140	Track Double	1992-02-21 15:02:00	0.00	1

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Datensätze verändern.

### Beispiel

- Hier wird das Preisgeld des Datensatzes mit der `id` 138 auf den Wert 1000 geändert. Das nachfolgende select zeigt, dass genau dieses Attribut verändert wurde.

```
SELECT S.id, S.name ,S.preisgeld, S.ist_mann  
FROM sportler S WHERE S.id>=138;
```

138	Tick Double	0.00	1
139	Trick Double	0.00	1
140	Track Double	0.00	1

```
UPDATE sportler SET preisgeld=1000 WHERE id=138;
```

138	Tick Double	1000.00	1
139	Trick Double	0.00	1
140	Track Double	0.00	1

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Mehrere Attribute in mehreren Datensätze verändern.

### Beispiel

- Es ist auch möglich, mehrere Datensätze und mehrere Attribute mit einem Befehl zu verändern.
- Hier wird das Preisgeld um 50 erhöht und der Status `ist_mann` auf `false` gesetzt.
- **Achtung:** Wird `WHERE` weggelassen, so werden *alle* Datensätze verändert. Manche Tools, etwa MySQL-Workbench oder DataGrip, können diesen Automatismus durch Einstellungen verhindern - Stichwort 'Unsafe Query' oder 'Safe Updates' oder ähnlich.

```
UPDATE sportler
SET preisgeld=preisgeld+50, ist_mann=false
WHERE id>=138;
```

id	name	preisgeld	ist_mann
138	Tick Double	1050.00	0
139	Trick Double	50.00	0
140	Track Double	50.00	0

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Datensätze ändern und dabei Subselects verwenden.

### Beispiel

- Wir betrachten, um es einfach zu halten, die `id` aus der Vorrunde als das max. zu gewinnende Preisgeld.
- Nun wird das Preisgeld auf den Wert gesetzt, der mit dem Subselect aus `'wettkampf'` ermittelt wird.
- Achtung: Liefert das Subselect mehrere Datensätze, so gibt es einen Fehler.  
Ist umgekehrt das Ergebnis `NULL`, so wird das Attribut auch auf `NULL` gesetzt.

```
SELECT max(id) as 'preisgeld'  
FROM wettkampf WHERE bezeichnung LIKE '%vorrunde%';
```

```
UPDATE sportler  
SET preisgeld = (  
    SELECT max(id) as 'preisgeld' FROM wettkampf  
    WHERE bezeichnung LIKE '%vorrunde%'  
) WHERE id=139;
```

ID	Name	Geburtsdatum	Preisgeld	Ist_Mann
139	Trick Double	1992-02-21 15:01:00	56.00	0

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Datensätze ändern und dabei Subselects verwenden, die sich auf die zu verändernde Tabelle, hier 'sportler', beziehen.

### Beispiel

- Im DBMS Oracle und MariaDB ist erster SQL-Befehl kein Problem, in manchen MySQL-Versionen schon.
- In der Alternative wird das eingebettete Subselect X von MySQL akzeptiert, jedoch ist unklar, ob die interne Optimierung diesen Workaround kompensieren kann.

```
UPDATE sportler
SET preisgeld = (
    SELECT preisgeld FROM sportler WHERE id=138
) WHERE id=140;
SELECT * FROM sportler WHERE id=140;
```

```
UPDATE sportler
SET preisgeld = 1+(
    SELECT preisgeld FROM
        (SELECT preisgeld FROM sportler WHERE id=138) X
) WHERE id=140;
```

Beispiel matse\_sport

## Daten anlegen, verändern, löschen

---

### Intention

- Datensätze löschen.

### Beispiel

- Hier werden alle Datensätze gelöscht, die der Bedingung entsprechen.
- **Achtung:** Wie bei UPDATE gilt, dass wenn WHERE weggelassen wird, *alle* Datensätze gelöscht werden.

```
DELETE FROM sportler WHERE id>130;
```

Beispiel matse\_sport

The screenshot shows a MySQL command-line interface. A warning message '102 !' is displayed above the input field. The user has typed the SQL command:

```
DELETE FROM sportler;
```

Below the command, a tooltip-like message reads: "Unsafe query: 'Delete' statement without 'where' clears all data in the table". At the bottom right of the interface, there are three buttons: "Execute", "Execute and Suppress", and a close button "X".

## Daten anlegen, verändern, löschen

---

### Aufgaben

#### 1. Wiederholung Modellierung

Stellen Sie sich vor, dass Sie an einem Institut der FH Aachen eine Position als Assistent antreten. Ihre erste Aufgabe besteht in der Datenreorganisation der Experimente, denn Ihrem Vorgänger wurde wegen seines chaotischen Arbeitsverhaltens und unauffindbarer Versuchsdaten gekündigt.

Sie stellen mit Schrecken fest, dass Dateien mit Versuchsdaten und Parametern zu verschiedenen Experimenten unsortiert und ohne erkennbares System auf dem Server "herumliegen". Sie wollen sich mit einer Datenbank einen ersten Überblick verschaffen und alle gefundenen Informationen ablegen.

Modellieren Sie einen Entwurf entsprechend nachfolgender Anforderungen in einem ER-Diagramm.

## Daten anlegen, verändern, löschen

---

### Aufgaben

#### 1. Fortsetzung: Anforderungen Modellierung

- ♦ Eine Tabelle für Experimente, in der Sie Informationen zum jeweiligen Experiment ablegen und dem Sie auch Dateien zuordnen können, erscheint Ihnen sinnvoll.
- ♦ Zu einem Experiment gibt es einen beliebigen beschreibenden Text, sowie die Information, wann zum letzten Mal an diesem Experiment gearbeitet wurde. Beispiel Experiment 'Kalte Fusion', letztmalig bearbeitet am '1.11.2014, 12:02:03'.
- ♦ Für die Zuordnung der zum Experiment gehörenden Dateien entscheiden Sie, dass Sie in dieser ersten Version in einer Entität nicht die Inhalte der Datei selbst, sondern nur den Pfad auf eine existierende Datei speichern wollen. Nützlich ist darüber hinaus, ebenfalls eine Charakterisierung des Inhalts, also eine beschreibende Information, zur Datei ablegen zu können.

## Daten anlegen, verändern, löschen

---

### Aufgaben

#### 1. Fortsetzung: Anforderungen Modellierung

- ♦ Da Sie sowohl Daten- als auch Parameterdateien erfassen möchten, speichern Sie in einem Typ die Art der Datei: eine '1' steht für eine Parameterdatei, eine '2' für eine Datendatei.

Beispiel: In der Parameterdatei (Typ '1'), zu finden unter 'c:/params/P1', befinden sich 'Temperatur und Angaben zum Druck'. Diese Datei gehört zum Experiment 'Kalte Fusion'.

Modellieren Sie einen Entwurf in einem ER-Diagramm.

**Sehen Sie sich bitte noch nicht die folgenden Aufgaben an!**

## Daten anlegen, verändern, löschen

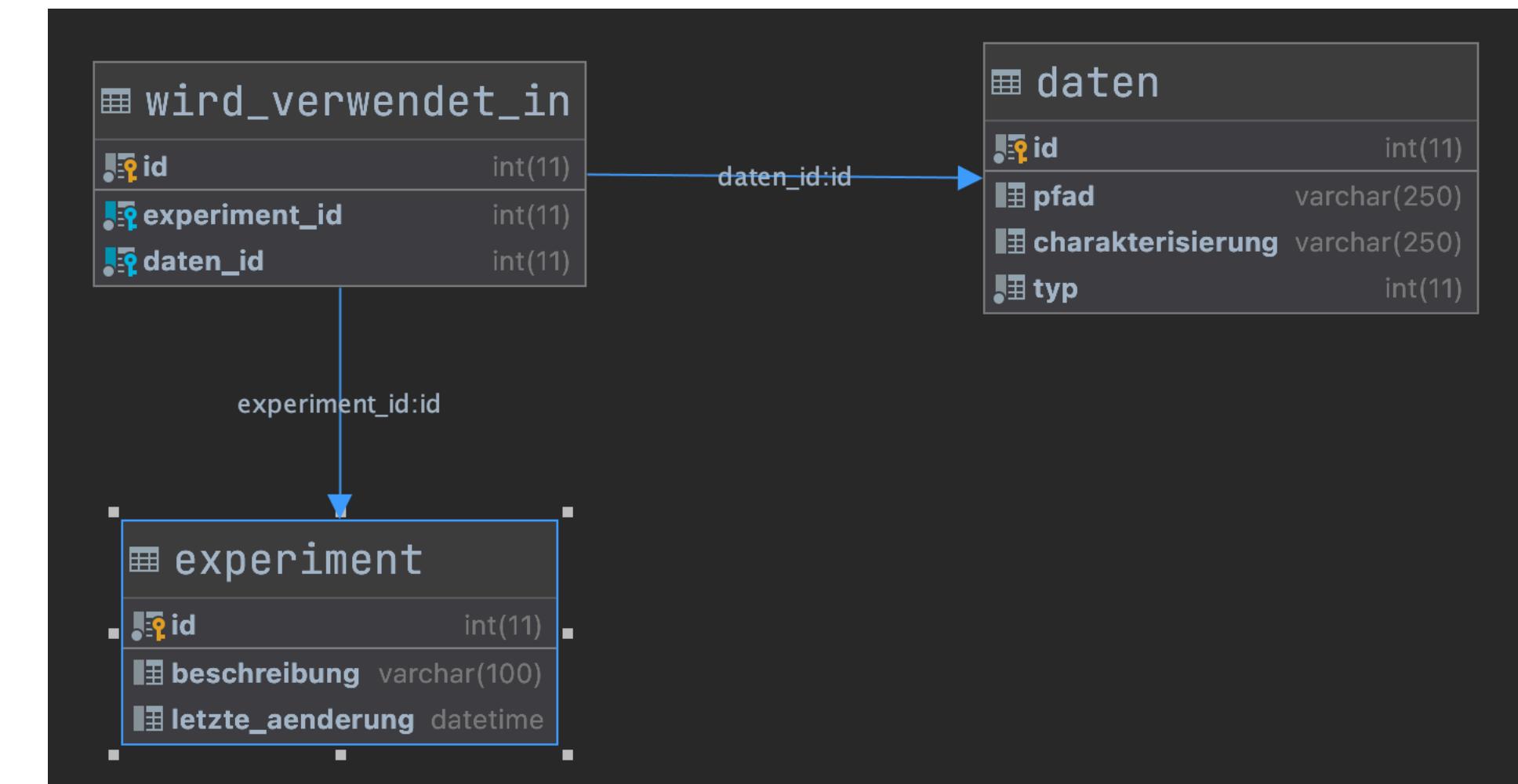
---

### Aufgaben

#### 2. SQL-Kommandos

Nebenstehend ist ein Entwurf zu sehen.  
Ihr ER-Diagramm sollte so oder so ähnlich aussehen.

Legen Sie das Schema `matse_experiment` und entsprechende Tabellen für die Entitätsarten `experiment`, `wird_verwendet_in` und `daten` an.



**Schreiben Sie hier und nachfolgend die SQL-Befehle erst zur Übung auf ein Papier und testen Sie sie erst dann in Ihrem DBMS.**

## Daten anlegen, verändern, löschen

---

### Aufgaben

#### 3. SQL-Kommandos

Fügen Sie folgende Daten per SQL-Kommando in die Tabellen ein:

id	beschreibung	letzte_aenderung
15	Kalte Fusion	2014-11-01 12:02:03
16	Heisse Fusion	2014-11-02 14:05:06
17	Knallgas	2014-10-03 17:08:09

Tabelle experiment

id	pfad	charakterisierung	typ
33	c:/params/P1	{T=1.3e8, P=1.4e6}	1
34	c:/params/P2	{v=0.5c}	1
35	c:/params/P3	{T=1.6e8, P=1.2e6}	1
42	c:/daten/D1a	{1/2}	2
43	c:/daten/D1b	{2/2}	2
44	c:/daten/D2	{1/1}	2

Tabelle daten

id	experiment_id	daten_id
1	15	33
2	15	34
3	15	42
4	15	43
5	16	34
6	16	35
7	16	44

Tabelle wird\_verwendet\_in

Starten Sie hier für ein paar Kommandos ebenfalls auf dem Papier.

## Daten anlegen, verändern, löschen

---

### Aufgaben

Nun wird ein weiterer Versuch zum Experiment 'Kalte Fusion' durchgeführt und es fallen neue Daten an bzw. ändern sich.

4. Ändern Sie das Datum des letzten Zugriffs des Experiments 'Kalte Fusion' auf den 5. Nov. 2014, 18:09 und 10 Sek.

Achtung: Sie müssen die `id` des Experiments 'Kalte Fusion' in dieser Aufgabe per Subselect suchen und dürfen nicht einfach die Bedingung `id=15` nutzen!  
In den nachfolgenden Aufgaben ist das dann in Ordnung.

5. Geben Sie die zu diesem Experiment bislang verfügbaren Datensätze aus. Zur Erinnerung: Datensätze haben den Typ 2, wogegen Parametersätze mit Typ 1 gekennzeichnet sind.

<code>id</code>	<code>pfad</code>	<code>charakterisierung</code>	<code>typ</code>
42	c:/daten/D1a	{1/2}	2
43	c:/daten/D1b	{2/2}	2

## Daten anlegen, verändern, löschen

---

### Aufgaben

6. Nutzen Sie die Ergebnisse aus Aufgabe 5, um daraus leicht veränderte neue Daten in daten, aber ohne einzelne `INSERT`-Befehle, zu generieren.

Gehen Sie ausnahmsweise davon aus, dass die neuen IDs sich um 3 von den alten unterscheiden. Es soll kein autoincrement genutzt werden. Hängen Sie die Version 2 ('\_v2') an die Charakterisierung an.

7. Analog zu Aufgabe 6 fügen Sie nun die passenden Entitäten in `wird_verwendet_in` hinzu, so dass die Daten aus Aufgabe 6 auch zum Experiment 'Kalte Fusion' gehören. Für die erforderlichen IDs wählen Sie, ausnahmsweise wie oben, einen beliebigen Offset, z.B. +5. Die neuen Daten Typ 2 in `wird_verwendet_in` sehen so aus.

id	pfad	charakterisierung	typ
45	c:/daten/D1a_V2	{1/2}	2
46	c:/daten/D1b_V2	{2/2}	2

id	experiment_id	daten_id
3	15	42
4	15	43
8	15	45
9	15	46

## Daten anlegen, verändern, löschen

---

### Aufgaben

8. Ermitteln Sie alle zum Experiment 'Kalte Fusion' gehörenden Dateien in dieser Form:

id	pfad	charakterisierung
33	c:/params/P1	{T=1.3e8, P=1.4e6}
34	c:/params/P2	{v=0.5c}
42	c:/daten/D1a	{1/2}
43	c:/daten/D1b	{2/2}
45	c:/daten/D1a_V2	{1/2}
46	c:/daten/D1b_V2	{2/2}

9. Da bei diesem Experiment doch ein Verfahrensfehler gefunden wurde, löschen Sie die Einträge mit den Versuchsdaten in daten und wird\_verwendet\_in. Lassen Sie das Experiment selber bestehen, wer weiß...

# UNIT 0X0A

## TRIGGER, STORED PROCEDURES

## Trigger, Stored Procedures

---

### Lernziel

- Stored Procedures und Trigger in Aktion gesehen haben - nur Kurzübersicht.

### Vorab - das Befehlsendezeichen ';'

- Um dem DBMS eine Definition einer Stored Procedure oder eines Triggers übergeben zu können, muss die Bedeutung des Befehlsendezeichens, das Semikolon ';', temporär geändert werden, da sonst ein ';' innerhalb der Procedure oder des Triggers zum Ende der Definition selber führen würde. Daher definiert man mittels DELIMITER ein eigenes Ersatzzeichen vor der Definition, nutzt dann das ';' innerhalb, beendet die Definition mit dem Ersatzzeichen und setzt am Ende das Befehlsendezeichen wieder zurück auf ';'.  
• Häufig werden '//' oder '\$\$' als Ersatzzeichen verwendet.

## Stored Procedures

---

### Intention

- Stored Procedures anlegen, aufrufen und löschen.

### Beispiel

- Nebenstehender CREATE PROCEDURE Befehl legt eine Stored Procedure namens ShowProducts an, die die Tabelle produkt abfragt.
- Aufgerufen wird eine Procedure mit CALL,
- und gelöscht, analog zu Tabellen, mit DROP.
- Achtung: in DataGrip muss der gesamte Block markiert und ausgeführt werden, damit die Befehlsendezeichen richtig gesetzt werden.

```
DELIMITER //  
CREATE PROCEDURE ShowProducts()  
BEGIN  
    SELECT * FROM produkt P;  
END //  
DELIMITER ;
```

```
CALL ShowProducts();
```

```
DROP PROCEDURE ShowProducts;
```

Beispiel matse\_mhist

## Stored Procedures

---

### Intention

- Stored Procedures mit Parametern anlegen.

### Beispiel

- Hier wird zunächst eine Stored Procedure namens ShowOneInProducts definiert, die einen Parameter `id` vom Typ `int` übergeben bekommt, um die Selektion einzuschränken.
- Ausblick: Es gibt nicht nur Stored Procedures sondern auch Stored Functions, die Resultate zurückliefern können. Ebenso gibt es weitere Parametertypen, aber das ist hier nicht weiter relevant.

```
DELIMITER //
CREATE PROCEDURE ShowOneInProducts(IN id int)
BEGIN
    SELECT P.* FROM produkt P where P.id=id;
END //
DELIMITER ;
```

Beispiel matse\_mhist

## Trigger

---

### Intention

- Trigger in Aktion verstehen.

### Beispiel

- Dieser CREATE TRIGGER Befehl legt einen Trigger before\_produkt\_update an, der vor einem Update auf produkt aufgerufen wird und für jede Änderung einen Eintrag in eine Log-Tabelle vornimmt, der den alten und den neuen Wert des Attributes bezeichnung, sowie einen Zeitstempel enthält.

```
DELIMITER $$  
CREATE TRIGGER before_produkt_update BEFORE UPDATE ON produkt  
FOR EACH ROW BEGIN  
    INSERT INTO log_update (table_name, last_text, new_text, last_update)  
    VALUES ('produkt', OLD.bezeichnung, NEW.bezeichnung, NOW());  
END$$  
DELIMITER ;
```

#	table_name	last_text	new_text	last_update
3	produkt	Frikadellen	Buletten	2021-01-12 13:14:33
4	produkt	Buletten	Frikadellen	2021-01-12 13:14:34

Tabelle log\_update

## Trigger, Stored Procedures

### Aufgaben

1. Legen Sie eine Stored Procedure Ihrer Wahl an, führen Sie sie aus und löschen Sie sie danach wieder.

# GESCHAFFT...

PROF. DR. RER. NAT. ALEXANDER VÖß

FH AACHEN  
UNIVERSITY OF APPLIED SCIENCES