R | R | Z | N
Regionales Rechenzentrum für Niedersachsen

Leibniz Universität Hannover

PostGre – SQL-Anweisungen in pgAdmin

S(tructured)Q(uery)L(anguage) ...

- ist eine eigenständige Sprache für relationale Datenbanken.
- bietet Befehle zum Bearbeiten von Daten und Erstellen von Objekten in relationalen Datenbanken.
- beschreibt eine Aktion wie zum Beispiel „Erstelle eine Tabelle“ oder „Alle Kunden aus Hannover“.
- ist standardisiert.
- wird von allen gängigen Datenbanksystemen mehr oder weniger unterstützt.

Manual

- PostgreSQL-Manual (englisch): http://www.postgresql.org/docs/manuals/
- PostgreSQL-Manual (deutsch): http://doc.postgres.de/

Beispieldatenbanken

- http://wiki.postgresql.org/wiki/Sample_Databases
- http://www.postgresqltutorial.com/postgresql-sample-database

Weitere Informationen

- http://www.schulserver.hessen.de/darmstadt/lichtenberg/SQLTutorial/
- http://sql.lernenhoch2.de/lernen/
- http://ssl.gymnasium-zwettl.ac.at/postgresql/html/tutorial.html
- http://sqlzoo.net/de/

… besteht aus ...

- DDL (Data Definition Language). Datenbanken, Tabellen etc. anlegen. Datenbankschema festlegen.
- DML (Data Manipulation Language). Daten in einer Datenbank lesen, ändern, einfügen oder löschen.
- DCL (Data Controlling Language). Administration von Datenbanken.

Menübefehle „Neue Datenbank”

- *Bearbeiten – Neues Objekt – Neue Datenbank*

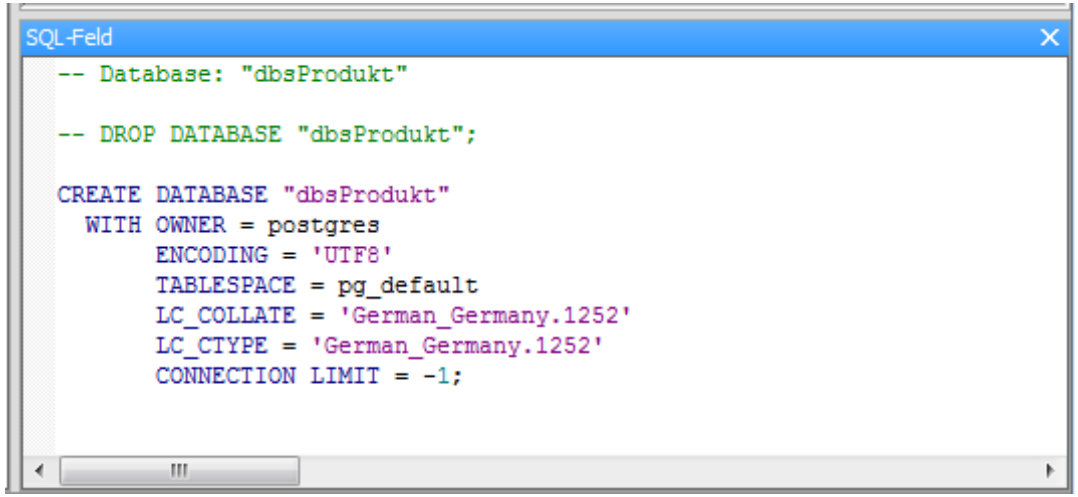
```
-- Database: "dbsProdukt"
-- DROP DATABASE "dbsProdukt";

CREATE DATABASE "dbsProdukt"
  WITH OWNER = postgres
       ENCODING = 'UTF8'
       TABLESPACE = pg_default
       LC_COLLATE = 'German_Germany.1252'
       LC_CTYPE = 'German_Germany.1252'
       CONNECTION LIMIT = -1;
```

SQL-Anweisungen ...

- Beginn mit einem SQL-Befehl.
- Semikolon als Ende-Zeichen einer Anweisung.
- Leerzeichen, Tabulatoren, neue Zeilen erhöhen die Lesbarkeit von Anweisungen.
- Groß- und Kleinschreibung bei Befehlen und Bezeichnern wird nicht beachtet.

SQL-Feld in pgAdmin

- Anzeige von SQL-Befehlen zu dem gewählten Objekt.
- Schreibschutz der Anweisungen.
- Kommentare werden standardmäßig grün angezeigt. SQL-Schlüsselwörter werden blau angezeigt.



Kommentare in SQL-Anweisungen

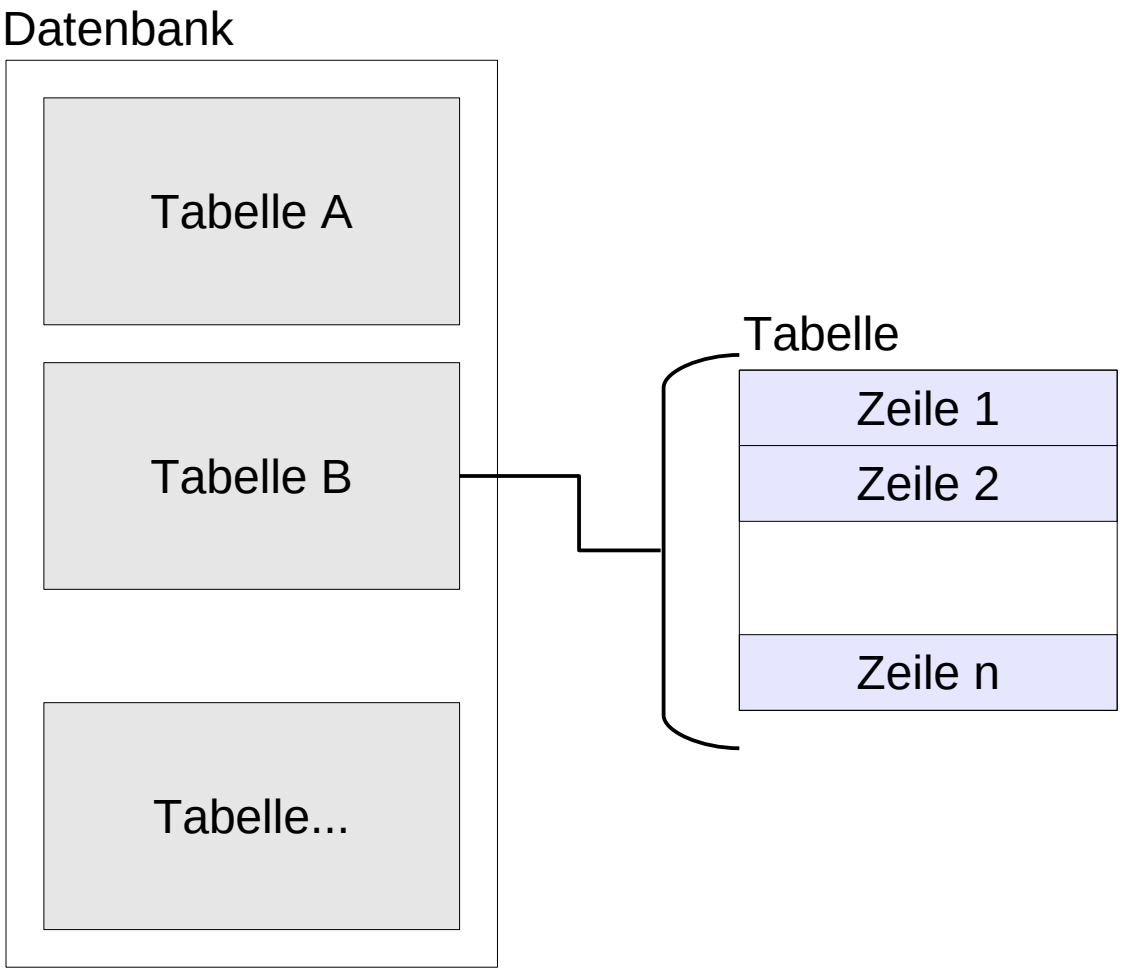
```
-- Database: "dbsProdukt"
-- DROP DATABASE "dbsProdukt";
```

- Der Kommentar beginnt mit zwei Bindestrichen und endet automatisch mit der Zeile.
- Aufgrund der besseren Lesbarkeit folgt dem Bindestrichpaar ein Leerzeichen.
- Beliebige Positionierung.

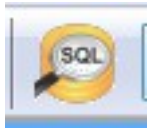
SQL-Anweisung „Neue Datenbank”

```
--  CREATE DATABASE [Datenbankname]
CREATE DATABASE "dbsProdukt"
  WITH OWNER = postgres                 --  Eigentümer der Datenbank
       ENCODING = 'UTF8'                -- Zeichencodierung
       TABLESPACE = pg_default          -- Speicherort der Datenbank
       -- Betrifft die Zeichen
       LC_COLLATE = 'German_Germany.1252' -- Sortierung
       LC_CTYPE = 'German_Germany.1252' --  Kategorisierung
       CONNECTION LIMIT = -1;           -- -1 beliebig viele Zugriffe
```

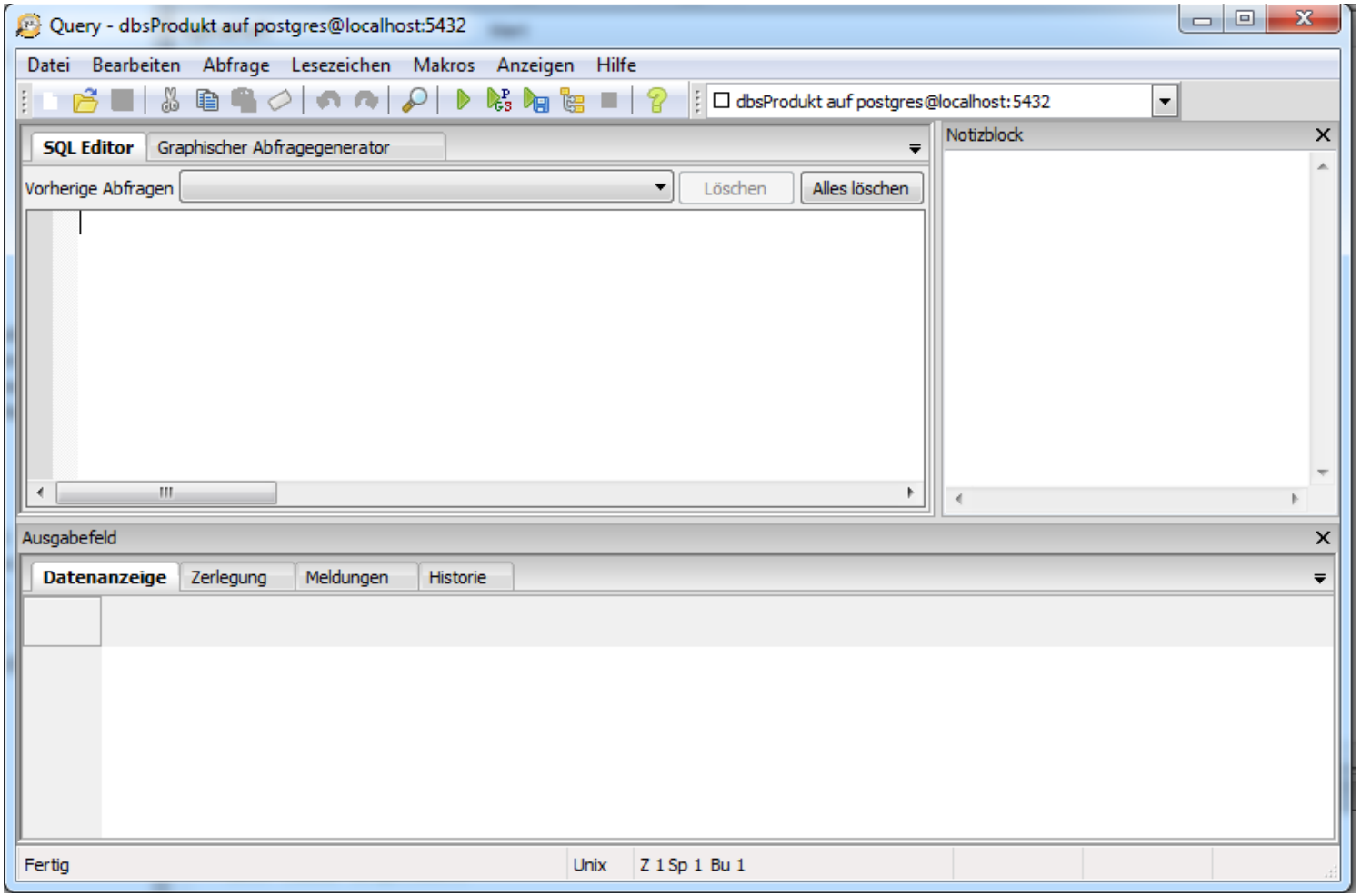
Aufbau einer Datenbank



SQL-Anweisung eingeben

- *Werkzeuge – Abfragewerkzeug*.
- Oder: *Beliebige SQL-Abfragen ausführen*  in der Symbolleiste.

Query


Query - dbsProdukt auf postgres@localhost:5432
Datei Bearbeiten Abfrage Lesezeichen Makros Anzeigen Hilfe
dbsProdukt auf postgres@localhost:5432
SQL Editor
Graphischer Abfragegenerator
Vorherige Abfragen
Löschen
Alles löschen
Notizblock
Ausgabefeld
Datenanzeige
Zerlegung
Meldungen
Historie
Fertig
Unix
Z 1 Sp 1 Bu 1

SQL-Editor

- Eingabe von SQL-Anweisungen.
- Ausführen von SQL-Anweisungen (*Abfrage – Ausführen*; F5). Das Ergebnis der SQL-Anweisung wird im Ausgabefeld; Registerkarte *Datenanzeige* angezeigt.
- Speichern von SQL-Anweisungen (*Datei – Speichern unter*).

Tabelle anlegen

```
--  CREATE TABLE [Tabllenname]
CREATE TABLE tblProdukt (
    --  Felder in der Tabelle
   --  Feldname                    Datentyp,
    produktName                    varchar(100),
    produktNr                      int,
    produktPreis                   numeric,
    aufgenommenAm                  timestamp
);
```

SQL-Befehle ...

- beschreiben eine Aktion. In diesem Beispiel CREATE TABLE erstellt eine neue Tabelle.
- werden häufig groß geschrieben.
- beginnen immer mit einem Buchstaben.

Auflistungen

- Die Listenelemente werden durch ein Kommata getrennt.
- Feldnamen werden dem Befehl CREATE TABLE als Liste hinzugefügt.
- Werte, die in einer Tabelle gespeichert werden, werden als Liste angegeben.
- Attribute eines Objekts werden beim Anlegen als Liste übergeben.

Anweisung ausführen

- *Abfrage* – *Ausführen* oder F5 im SQL-Editor.
- Im Ausgabefeld wird das Ergebnis der Ausführung angezeigt. Die Registerkarte *Meldungen* zeigt eventuelle Fehler oder die korrekte Ausführung an. Ausgewählte Daten werden auf der Registerkarte *Datenanzeige* angezeigt.
- Bei der Erstellung von Objekten, muss anschließend das Eltern-Objekt aufgefrischt werden (rechter Mausklick; *Auffrischen*).

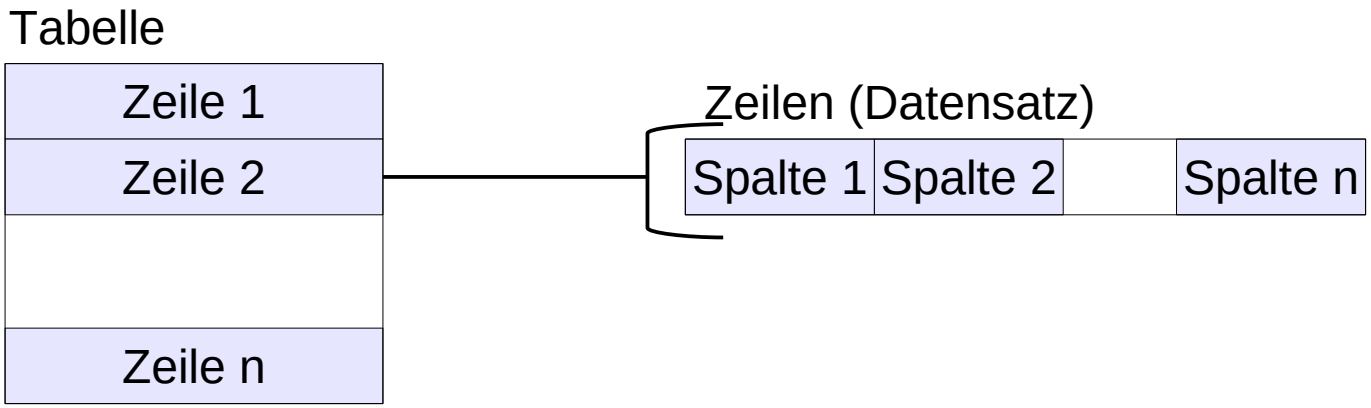
Anweisungen speichern und erneut ausführen

- *Datei – Speichern* unter im SQL-Editor.
- *Datei – öffnen* zeigt eine gespeicherte Anweisung im SQL-Editor an.
- F5 führt die Anweisung aus.

Bezeichner in SQL-Anweisungen

- sind Namen, die vom Benutzer für Tabellen, Datenfelder etc. definiert werden.
- in dem Beispiel sind tblProdukt, produktName etc.
- beginnen immer mit einem Buchstaben.
- können Buchstaben (A...Z, a...Z), Zahlen (0...9) und den Unterstrich enthalten.
- sind maximal 63 Zeichen lang.

Aufbau von Tabellen



Datensätze

- Die Anzahl der Datensätze ist beliebig.
- Die Datensätze werden unsortiert abgelegt.

produktName	produktNr	produktPreis	aufgenommenAm
Kaffee „Arabica”	1	5,99 €	20.11.2013 09:00
Tee „Darjeeling”	2	4,99 €	21.11.2013 10:23
Kaffee „Kenia”	3	7,99 €	21.11.2013 12:05

Spalten in einer Tabelle

- Identifizierung mit Hilfe eines eindeutigen Namens.
- Der Datentyp der Spalte beschreibt die Art des Inhaltes.
- Die Anzahl der Spalten und deren Reihenfolge ist fest.
- Die maximale Anzahl der Spalten liegt zwischen 250 und 1600.

tblProdukt
produktName : String produktNr : Integer produktPreis : Currency aufgenommenAm : Date / Time

Feldnamen ...

- sind eine eindeutige Bezeichnung für eine Spalte einer Tabelle.
- identifizieren eindeutig ein Datenfeld in einer Tabelle.
- entsprechen häufig der Bezeichnung des Attributes des beschriebenen Objekts in der realen Welt.
- sind frei wählbar.

Der Datentyp eines Feldes ...

- beschreibt die Art der Daten in einem Feld.
- legt Regeln für die Verwendung der Daten fest.
- legt den Speicherbedarf der Daten fest.
- ergibt sich häufig aus der Funktion und / oder Nutzung des, in dem Feld zu speichernden Wert.
- in PostgreSQL http://www.postgresql.org/docs/9.3/static/datatype.html.

Zeichenketten / Text

- Alle Zeichen der Tastatur in Abhängigkeit des Eingabegebietsschema.
- Speicherung von Telefonnummern, Postleitzahlen etc.
- Standardlänge 1.

Möglichkeiten

Datentyp	Beschreibung
char(N)	Zeichenkette von N Zeichen
varchar(N)	In Abhängigkeit der Länge der Zeichenkette N = maximale Zeichenkettenlänge

Zahlenwerte

- Mathematische Berechnungen
- Ganzzahl. Nachkommastellen werden automatisch entfernt.
- Dezimalzahlen sind Nährungswerte. Das Dezimaltrennzeichen ist von dem Eingabegebietsschema des Rechners abhängig. Implemtentierung nach dem IEEE Standard 754 für binäre Fließkommazahlen.

Ganzzahlen

Datentyp	Zahlenraum
smallint	-32768 bis +32767
int	-2147483648 bis +2147483647
bigint	-9223372036854775808 bis +9223372036854775807

Dezimalzahlen

Datentyp	Beschreibung
double precision	15 Stellen nach dem Komma genau
real	6 Stellen nach dem Komma genau
numeric(precision, scale)	131072 Stellen vor dem Komma und 16383 Stellen nach dem Komma precision = Genauigkeit scale = Anzahl der Nachkommastellen Standardwert von 0
money	-92233720368547758,08 bis +92233720368547758,07

Auto-Inkrement

Datentyp	Beschreibung
smallserial	1 bis 32767
serial	1 bis 2147483647
bigserial	1 bis 9223372036854775807

Datumswerte und Uhrzeiten

- Datumswerte basieren auf den gregorianischen Kalender.
- Abhängig vom genutzten SQL-Dialekt und das gewählte Eingabegebietsschema.

Möglichkeiten

Datentyp	Beschreibung
date	Jahr, Monat, Tag.
time	Stunde, Minute, Sekunde.
timestamp	Zeitstempel. Datum und Uhrzeit.

Standardwerte

```
CREATE TABLE tblProdukt (
    produktName             varchar(100),
    produktNr               int,
    produktPreis            numeric DEFAULT 2.00,
    aufgenommenAm           timestamp
);
```

Konstante Werte in SQL

- Strings / Zeichenketten: 'Dies ist eine Zeichenkette'.
- Integer / Ganzzahlen: int8 '400' oder 400.
- Dezimalzahlen: float4 '2.00' oder 2.0. Als Dezimaltrennzeichen wird der Punkt genutzt.
- Datums- und Zeitwerte: '2013-11-22'. Datumswerte werden in dem Format „yyyy-mm-dd” und Zeitwerte in dem Format „hh:mm:ss” angegeben

Aktueller „Zeitstempel“ als Standardwert

```
CREATE TABLE tblProdukt (
    produktName          varchar(100),
    produktNr            int,
    produktPreis         numeric,
    aufgenommenAm        timestamp
                         DEFAULT CURRENT_TIMESTAMP
);
```

NULL

```
CREATE TABLE tblProdukt (
    produktName          varchar(100) NOT NULL,
    produktNr            int,
    produktPreis         numeric,
    aufgenommenAm        timestamp
);
```

- Das Datenfeld darf keinen undefinierten Inhalt enthalten.
- Das Datenfeld muss gefüllt werden.

Tabelle löschen

```
DROP TABLE tblProdukt;
```

- Die Tabelle wird ohne Nachfrage gelöscht!

Primärschlüssel

- Identifizierung eines Datensatzes.
- Setzen von künstlichen Attributen. Das Produkt existiert auch ohne Produktnummer.
- Verknüpfung von Tabellen.

… in SQL

```
CREATE TABLE tblProdukt (
    produktName             varchar(100) NOT NULL,
    produktNr               int UNIQUE NOT NULL,
    aufgenommenAm           timestamp
);
```

```
CREATE TABLE tblProdukt (
    produktName             varchar(100) NOT NULL,
    produktNr               int PRIMARY KEY,
    aufgenommenAm           timestamp
);
```

Zusammengesetzte Primärschlüssel

```
CREATE TABLE tblProdukt (
    produktName             varchar(100) NOT NULL,
    produktNr               int,
    aufgenommenAm           timestamp,
    PRIMARY KEY (produktNr, aufgenommenAm)
);
```

Fremdschlüssel

- Der Wert identifiziert einen Datensatz in einer Tabelle.
- Der Fremdschlüssel und der dazugehörige Primärschlüssel haben den gleichen Datentyp.

… in SQL

```
CREATE TABLE tblProdukt (
    produktName             varchar(100) NOT NULL,
    produktNr               int PRIMARY KEY,
    aufgenommenAm           timestamp
);
```

```
CREATE TABLE tblLager (
    IDLager                 int PRIMARY KEY
    produktNr               int REFERENCES
                            tblProdukt(produktNr),
    bestandAenderung        int
);
```

Weitere Möglichkeit

```
CREATE TABLE tblLager (
    IDLager                 serial,
    FIDProduktNr            int,
    Entnahme                int
    PRIMARY KEY (IDLager),
    FOREIGN KEY (FIDProduktNr)
                    REFERENECES tblProdukt (produktNr)
);
```

Daten einfügen

```
INSERT INTO tblProdukt
VALUES
(
    'Kaffee Arabica',
    1,
    5.99,
    '2013-11-20 09:00'
);
```

Daten aus einer Textdatei einfügen

```
COPY tblProdukt FROM '/pfad/datei.txt';
```

- Der Pfad wird aus Sicht des Servers angegeben.
- Der Server muss die Rechte zum Lesen der Datei besitzen. Andernfalls wird ein Fehler angezeigt.
- Es können Textdateien oder Dateien im CVS-Format kopiert werden.
- Pro Zeile in der Textdatei wird ein Datensatz in der Tabelle angelegt.

Daten aus einer Textdatei einfügen

```
COPY tblProdukt FROM '/pfad/datei.txt';
```

- Der Pfad wird aus Sicht des Servers angegeben.
- Der Server muss die Rechte zum Lesen der Datei besitzen. Andernfalls wird ein Fehler angezeigt.
- Es können Textdateien oder Dateien im CVS-Format kopiert werden.
- Pro Zeile in der Textdatei wird ein Datensatz in der Tabelle angelegt.

Beispiel: Textdatei

Kaffee Arabica	1	5.99	2013-11-20 09:00
Tee Darjeeling	2	4.99	2013-11-21 10:23
Kaffee Kenia	3	7.99	2013-11-21 12:05

```
COPY tblProdukt FROM 'D:\temp\produkte.txt'
```

- Trennung durch Tabulatoren.
- Als Dezimaltrennzeichen wird der Punkt genutzt.
- Datums- und Zeitwerte sollten wie in SQL geschrieben werden.

Beispiel: CSV-Datei

Kaffee Arabica,1,5.99,2013-11-20 09:00

Tee \"Darjeeling\",2,4.99,2013-11-21 10:23

Kaffee Kenia,3,7.99,2013-11-21 12:05

```
COPY tblProdukt
(produktName, produktNr, produktPreis, aufgenommenAM)
FROM 'D:\temp\produkte.csv'
WITH DELIMITER ',';
```

Beispiel: CSV-Datei

```
COPY tblProdukt
(produktName, produktNr, produktPreis, aufgenommenAM)
FROM 'D:\temp\produkte.csv'
WITH DELIMITER ',';
```

- Dem Tabellennamen folgen in Klammern die Felder, die von der Datei gefüllt werden. Alle anderen Felder nutzen den vorgegebenen Standardwert oder sind leer.
- Das Trennzeichen zwischen den Daten wird angegeben (WITH DELIMITER).

Escape-Sequenzen ...

- beginnen immer mit einem Backslash \.
- sind nicht druckbare Zeichen eines Zeichensatzes.
- maskieren Zeichen, die in SQL-Anweisungen genutzt werden.

Sequenz	Erläuterung
\b	Backspace. Einen Schritt zurück.
\f	Formfeed. Seitenvorschub
\n	New Line. Neue Zeile.
\r	Carriage Return. Wagenrücklauf.
\t	Tabulator
\\	Backslash
\'	Apostroph
\"	Anführungszeichen

Auswahlabfragen ...

- sortieren Informationen aus ein oder mehreren Tabellen.
- filtern Informationen in Abhängigkeit von bestimmten Kriterien.
- stellen Informationen neu zusammen.
- berechnen Daten aus vorhandenen Informationen.
- liefern eine leere Tabelle zurück, wenn kein Datensatz die Bedingung erfüllt.

… in SQL

```
SELECT
*
FROM Tabelle
WHERE Kriterium
ORDER BY Tabelle.Feld;
```

```
SELECT
Tabelle.Feld, Tabelle.Feld, Tabelle.Feld
FROM Tabelle
WHERE Kriterium
ORDER BY Tabelle.Feld;
```

Alle Felder aus einer Tabelle

```
SELECT * FROM film;
```

- Wähle (SELECT) alle Felder (*) aus (FROM) ....
- Das Sternchen ist ein Platzhalter für alle Felder einer Tabelle.
- Dem Schlüsselwort FROM folgt der Name der Tabelle.

Felder aus einer Tabelle

```
SELECT
title, description, length
FROM film;
```

- Dem Befehl SELECT folgt eine Auflistung der gewünschten Felder einer Tabelle. Es muss mindestens ein Feld oder das Sternchen angegeben werden.
- Die einzelnen Felder werden durch ein Kommata getrennt.
- Das erste Feld in der Auflistung wird auch als erste Spalte angezeigt.

Felder aus einer Tabelle

```
SELECT
film.title, film.description, film.length
FROM film;
```

- Der Punkt verbindet ein Tabellennamen mit einem darin definierten Feldnamen.
- Die Angabe des Tabellennamens wird nicht benötigt, wenn der Feldname eindeutig einer Tabelle zugeordnet werden kann.

Daten sortieren

```
SELECT
last_name, first_name,  email
FROM customer
ORDER BY last_name, first_name;
```

- Dem Befehl ORDER BY folgt eine Liste von Feldern nach denen die Daten sortiert werden.
- Standardmäßig werden die Daten aufsteigend sortiert (ASC).
- In diesem Beispiel werden die Daten zuerst nach dem Nachnamen aufsteigend sortiert und anschließend nach dem Vornamen.

Daten aufsteigend sortieren

```
SELECT
film.title, film.description, film.length
FROM film
ORDER BY film.length ASC;
```

Daten absteigend sortieren

```
SELECT
film.title, film.description, film.length
FROM film
ORDER BY film.length DESC;
```

Doppelte Datensätze ausblenden

```
SELECT DISTINCT
release_year
FROM film
ORDER BY release_year DESC;
```

- Jahreszahlen werden nur einmal angezeigt, egal wie oft die Daten in der Tabelle angezeigt werden.
- Aus welchen Jahren liegen Filme vor?

Daten filtern

```
SELECT first_name, last_name
FROM staff
WHERE (active = TRUE)
ORDER BY last_name;
```

- Dem Befehl WHERE folgen eine oder mehrere Bedingungen.
- Die Datensätze werden mit Hilfe der Bedingung gefiltert.

Bedingungen

- Überprüfung von Daten in einem Feld.
- Ausdrücke, die einen boolschen (True, False) Wert zurückliefern.
- Mit Hilfe von Vergleichsoperatoren wird ein Wert überprüft.

Vergleichsoperatoren

ist ...	Operator
gleich	=
ungleich	<>
kleiner	<
kleiner gleich	<=
größer	>
größer gleich	>=

Überprüfung von Ganzzahlen

```
SELECT title, length
FROM film
WHERE (length > 60)
ORDER BY length;
```

Überprüfung von Dezimalzahlen

```
SELECT customer_id, amount
FROM payment
WHERE (amount > 5.5)
ORDER BY customer_id;
```

- Als Dezimaltrennzeichen wird ein Punkt genutzt.
- Es werden Nährungswerte überprüft!

Liste von Werten

```
SELECT title, rental_rate
FROM film
WHERE (rental_rate IN (0.99, 2.99))
ORDER BY title;
```

- Einer Werte in der Liste muss zutreffen.

Überprüfung von Datums- und Zeitwerten

```
SELECT *
FROM payment
WHERE (payment_date = '2007-03-01')
ORDER BY customer_id;
```

- Datums- und Zeitwerte werden in Apostrophs eingeschlossen.
- Konstante Datumswerte: YYYY-MM-DD.

Zwischen … und ...

```
SELECT *
FROM payment
WHERE (payment_date BETWEEN '2007-03-01' AND '2007-03-31')
ORDER BY customer_id;
```

- Zwischen [min] und [max].
- Zwischen [Beginn Datum / Uhrzeit] und [Ende Datum / Uhrzeit].

Überprüfung von boolschen Werten

```
SELECT first_name, last_name
FROM staff
WHERE (active IS TRUE)
ORDER BY last_name;
```

- IS TRUE. Ist das Feld gesetzt / an?
- IS FALSE. Ist das Feld nicht gesetzt / aus?

Überprüfung von Text

```
SELECT last_name, first_name
FROM customer
WHERE (last_Name = 'May')
ORDER BY last_name;
```

- Konstante Zeichenketten werden in Apostrophs eingeschlossen.

Arbeiten mit Suchmustern

```
SELECT last_Name, first_Name, email
FROM customer
WHERE (email LIKE '%.%@%')
ORDER BY last_name;
```

- Platzhalter %: Beliebige Anzahl von Zeichen.
- Platzhalter _: Ein beliebiges Zeichen.
- Die Platzhalter können an beliebiger Position in beliebiger Anzahl in einem Suchmuster vorkommen.
- Beachtung von Groß- und Kleinschreibung.

Suche nach leeren Zeichenfolgen

```
SELECT *
FROM address
WHERE (postal_code LIKE '')
ORDER BY city_id;
```

- Das Feld ist leer, aber hat einen definierten Wert.

Null-Werte abfragen

```
SELECT *
FROM address
WHERE (postal_code IS NULL)
ORDER BY city_id;
```

- Das Feld hat einen undefinierten Wert.

Not-Operator

```
SELECT *
FROM address
WHERE (postal_code NOT LIKE '')
ORDER BY city_id;
```

```
SELECT *
FROM address
WHERE (postal_code IS NOT NULL)
ORDER BY city_id;
```

a	Not a
True	False
False	True
Null	Null

AND-Operator

```
SELECT amount, customer_id
FROM payment
WHERE ((amount > 3.0) AND (amount < 4.0))
ORDER BY customer_id;
```

a	b	a AND b
True	True	True
True	False	False
False	True	False
False	False	False
True	Null	Null
False	Null	False

Or-Operator

```
SELECT amount, customer_id
FROM payment
WHERE ((customer_id = 5) OR (customer_id = 6))
ORDER BY customer_id;
```

a	b	a Or b
True	True	True
True	False	True
False	True	True
False	False	False
True	Null	True
False	Null	Null

Daten löschen

```
DELETE
FROM city
WHERE country_id = 20;
```

- Daten können nur kaskadierend gelöscht werden.
- Zuerst werden die Daten in den Tabellen gelöscht, in dem der Primärschlüssel als Fremdschlüssel genutzt wird. Anschließend können Datensätze mit dem Primärschlüssel gelöscht werden.

Daten ändern

```
UPDATE [Tabelle]
SET [Feldname] = [neuer Wert], [Feldname] = [neuer Wert]
WHERE [Bedingung];
```

```
UPDATE customer
SET active = 0, activebool = FALSE
WHERE store_id = 2;
```

```
UPDATE payment
SET amount = amount + 1
WHERE amount < 1;
```

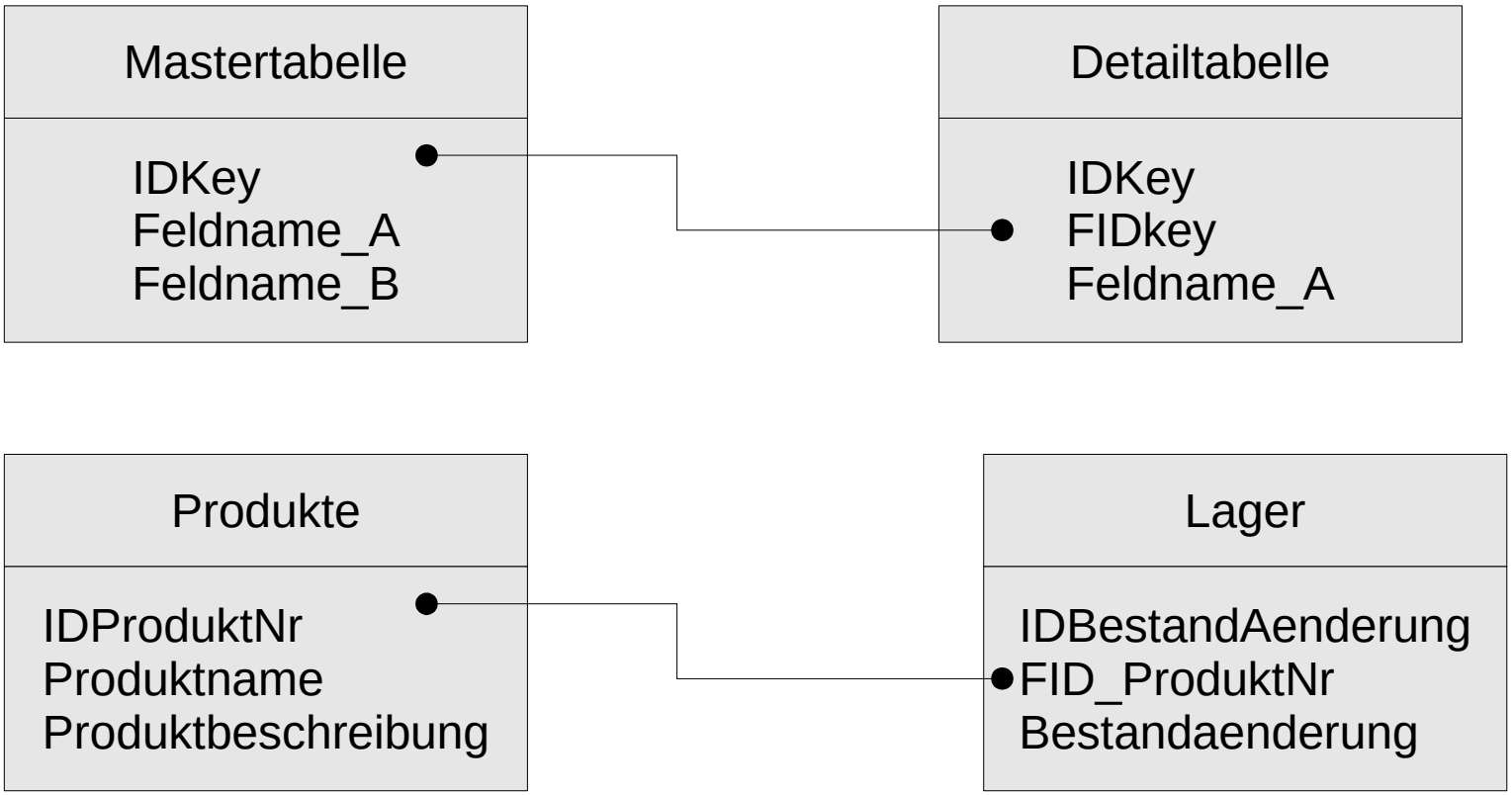
Mathematische Operatoren

Operator	Beschreibung	Beispiel
+	Addition	2 + 3 = 5
-	Subtraktion	2 – 3 = -1
*	Multiplikation	2 * 3 = 6
/	Division	5 / 2 = 2.5
%	Modula	5 % 2 = 1
^	Expontential	2^3 = 8
@	Absolutwert	@-5 = 5

Relationen ...

- beschreiben Beziehungen zwischen Tabellen.
- werden mit Hilfe von Primärschlüssel und Fremdschlüssel zwischen den Tabellen gebildet.
- beschreiben Hierarchien zwischen Tabellen.

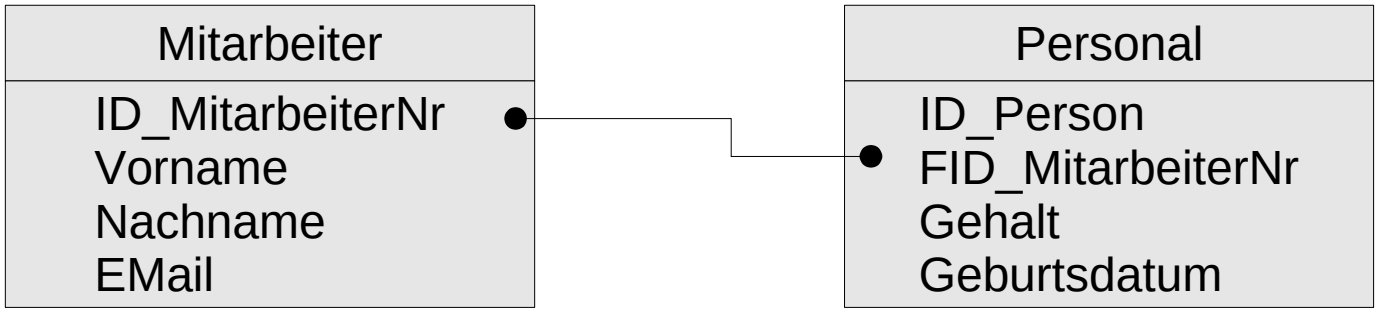
Beispiel


Mastertabelle
IDKey
Feldname_A
Feldname_B
Detailtabelle
IDKey
FIDkey
Feldname_A
Produkte
IDProduktNr
Produktname
Produktbeschreibung
Lager
IDBestandAenderung
FID_ProduktNr
Bestandaenderung

Master- und Detailtabelle

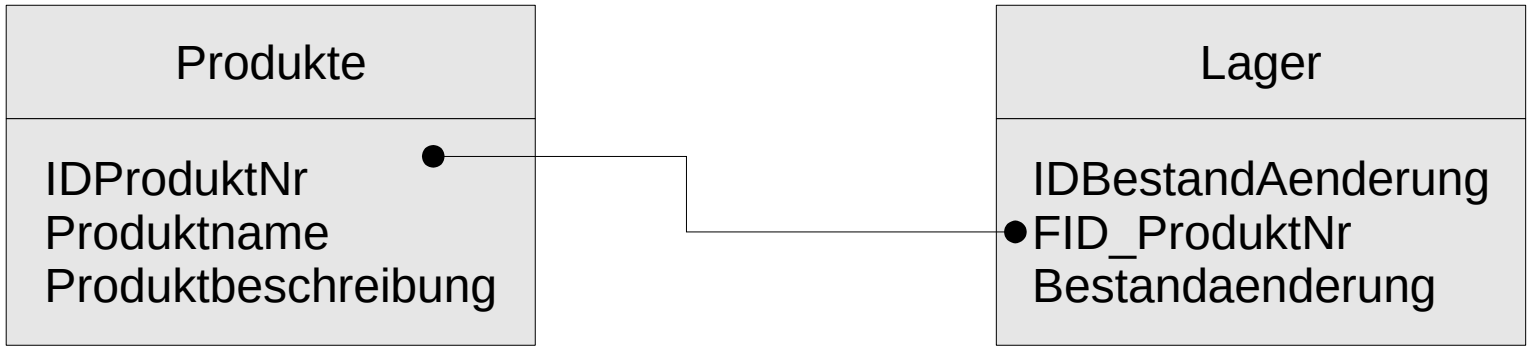
- Eine Mastertabelle beschreibt ein Objekt allgemein. Die Mastertabelle befindet sich in der Hierarchie oberhalb einer anderen Tabelle.
- Eine Detailtabelle beschreibt ein Detail-Aspekt eines Objekts. Detailtabellen besitzen einen Fremdschlüssel aus einer anderen Tabelle.
- Eine Mastertabelle kann eine Detailtabelle für ein bestimmtes Objekt sein.

1:1-Relation



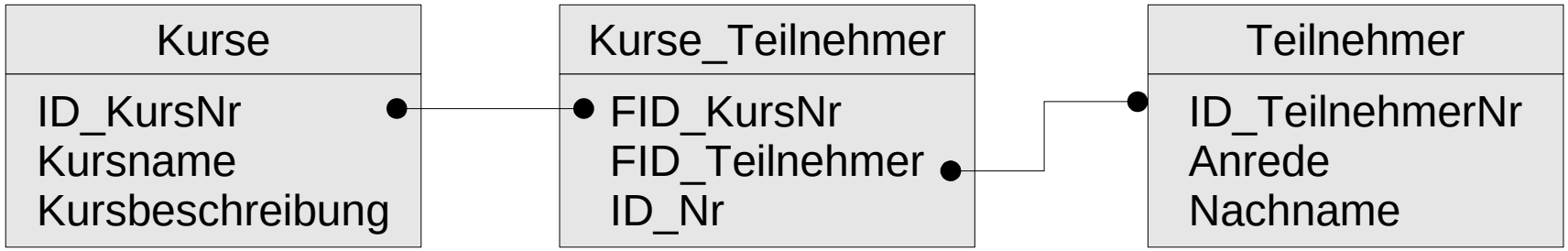
- Jedem Datensatz aus der Mastertabelle kann exakt ein Datensatz in der Detailtabelle zugeordnet werden.
- Nutzung bei Tabellen mit sehr vielen Spalten.
- Informationen werden in Abhängigkeit ihrer Nutzung gespeichert.

1:n-Relation



- Jedem Datensatz aus der Mastertabelle können beliebig viele Datensätze aus der Detailtabelle zugeordnet werden.
- Häufigst vorkommende Relation.

m:n-Relation



- Beliebig viele Datensätze können mit Datensätzen aus einer anderen Tabelle verknüpft werden.
- Die Relation wird in einer Pseudo-Tabelle abgebildet.

Inner-Join-Verknüpfung

```
SELECT
customer.last_name, customer.first_name,
payment.amount
FROM customer
INNER JOIN payment
ON customer.customer_id = payment.customer_id;
```

- Aktion: Kunden und deren Bezahlung.

Beispiel

ID	Erdteil
1	Europa
2	Asien
3	Afrika
4	Amerika

Erdteil	Land
Europa	Belgien
Europa	Polen
Afrika	Algier
Asien	Indien

ID	Erdteil	Land
1	1	Belgien
2	1	Polen
3	3	Algier
4	2	Indien

Left-Join-Verknüpfung

```
SELECT
customer.last_name, customer.first_name,
payment.amount
FROM customer
LEFT JOIN payment
ON customer.customer_id = payment.customer_id;
```

- Aktion: Alle Kunden und falls vorhanden, deren Bezahlung.

Beispiel

ID	Erdteil
1	Europa
2	Asien
3	Afrika
4	Amerika

Erdteil	Land
Europa	Belgien
Europa	Polen
Afrika	Algier
Asien	Indien
Amerika	

ID	Erdteil	Land
1	1	Belgien
2	1	Polen
3	3	Algier
4	2	Indien

Right-Join-Verknüpfung

```
SELECT
language.name,
film.title
FROM film
RIGHT JOIN language
ON film.language_id = language.language_id;
```

- Aktion: Alle Sprachen und falls vorhanden, die dazugehörigen Filme.

Beispiel

ID	Erdteil
1	Europa
2	Asien
3	Afrika
4	Amerika

Erdteil	Land
Europa	Belgien
Europa	Polen
Afrika	Algier
Asien	Indien

ID	Erdteil	Land
1	1	Belgien
2	1	Polen
3	3	Algier
4	2	Indien

… und sortieren

```
SELECT
language.name,
film.title
FROM film
RIGHT JOIN language
ON film.language_id = language.language_id
ORDER BY language.name, film.title;
```

Daten filtern

```
SELECT
language.name,
film.title, film.length
FROM film
RIGHT JOIN language
ON film.language_id = language.language_id
WHERE film.length > 100
ORDER BY language.name, film.length;
```

1 : n : 1 -Relation

```
SELECT
category.name,  film.title

FROM film

INNER JOIN film_category
ON film.film_id = film_category.film_id

INNER JOIN category
ON film_category.category_id = category.category_id;
```

Aggregatfunktionen ...

- fassen das Ergebnis einer SQL-Anweisung zusammen.
- gruppieren alle Datenfelder in einer Abfrage.
- komprimieren Datensätze auf die angegebenen Datenfelder.

… in SQL

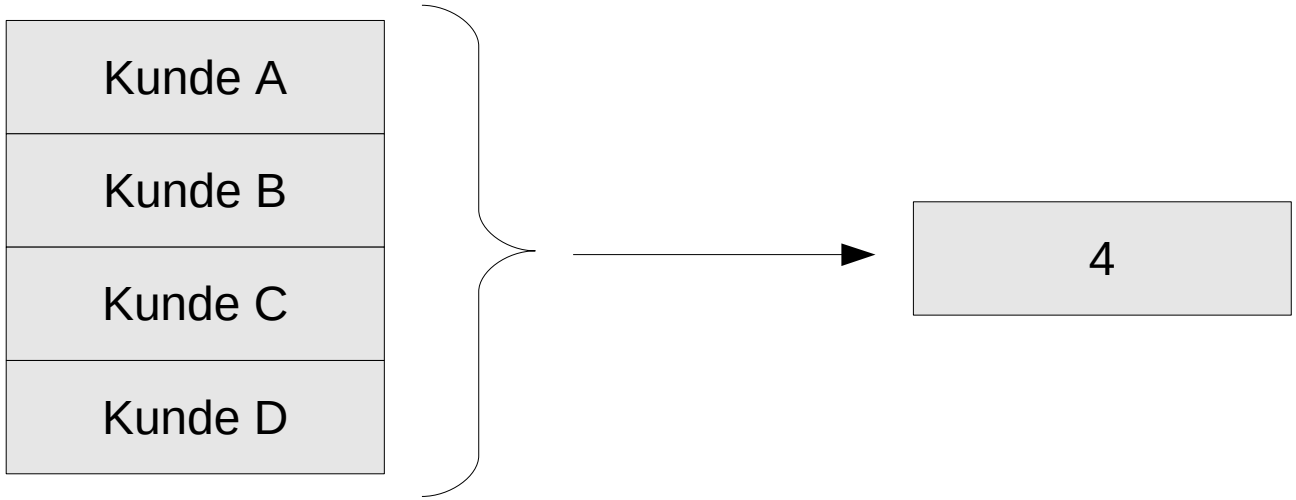
Aggregatfunktion	Erläuterung
Sum([Datenfeld])	Summe einer Spalte.
Avg([Datenfeld])	Durchschnitt einer Spalte.
Min([Datenfeld])	Kleinster Wert einer Spalte.
Max([Datenfeld])	Größter Wert einer Spalte.
Count([Datenfeld])	Anzahl der Datensätze. Leere Datenfelder werden nicht berücksichtigt.
Count(*)	Alle Datensätze zählen. Undefinierte Datenfelder werden nicht berücksichtigt.

Anzahl der Datensätze

```
SELECT
COUNT(*) AS customerPayCount
FROM customer
INNER JOIN payment
ON customer.customer_id = payment.customer_id;
```

- Die Anzahl der Zeilen wird ausgegeben.
- Mit Hilfe von AS wird dem berechneten Feld ein neuer Name zugewiesen.

Grafische Darstellung

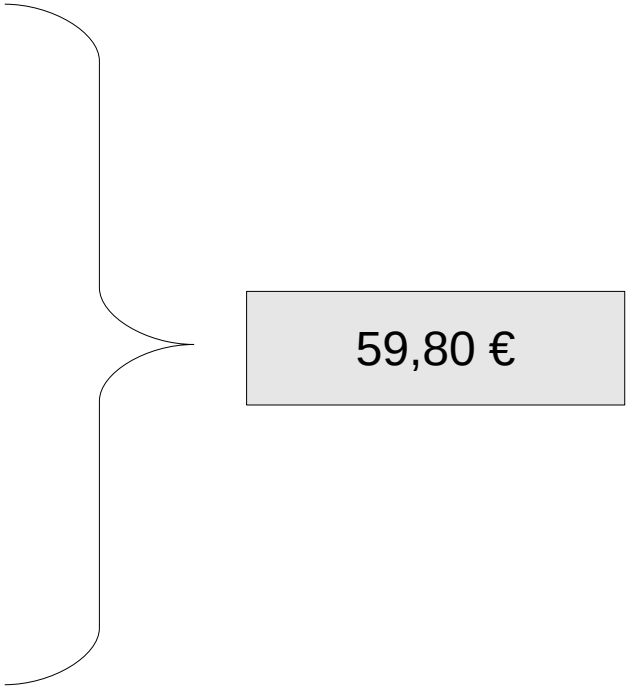


Summe von Feld x

```
SELECT
SUM(amount) AS allPayment
FROM customer
INNER JOIN payment
ON customer.customer_id = payment.customer_id;
```

Grafische Darstellung

6,20 €	1
7,80 €	1
6,20 €	2
6,80 €	2
4,20 €	1
7,80 €	2



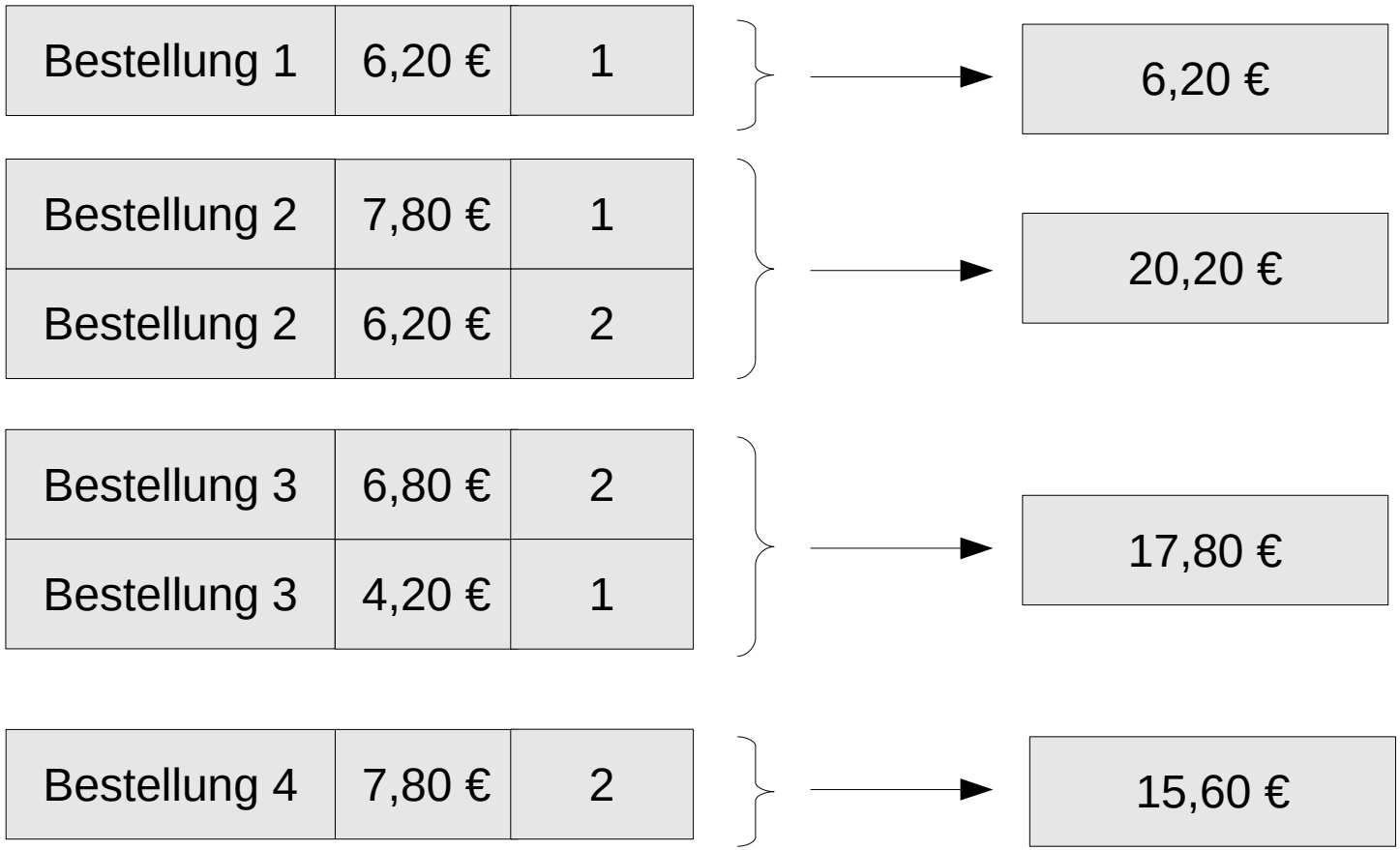
Felder gruppieren

```
SELECT
customer.last_Name,
SUM(amount) AS PaymentForCustomer
FROM customer
INNER JOIN payment
ON customer.customer_id = payment.customer_id

GROUP BY customer.last_Name

ORDER BY customer.last_Name;
```

Hinweis

- Für jedes Feld, welches in der SQL-Anweisung angegeben ist und nicht mit einer Aggregatfunktion wie Sum(), Count(), usw. berechnet wird, muss mit der Group-By-Anweisung zusammengefasst werden.
- Null-Werte werden in der Group-By-Anweisung zusammengefasst.

Grafische Darstellung



Daten filtern

```
SELECT
customer.last_Name,
SUM(amount) AS PaymentForCustomer
FROM customer
INNER JOIN payment ON customer.customer_id = payment.customer_id

WHERE customer.last_Name LIKE 'A%'

GROUP BY customer.last_Name

ORDER BY customer.last_Name;
```

Having-Klausel

```
SELECT
customer.last_Name,
SUM(amount) AS PaymentForCustomer
FROM customer
INNER JOIN payment
ON customer.customer_id = payment.customer_id

GROUP BY customer.last_Name
HAVING SUM(amount) > 100

ORDER BY customer.last_Name;
```

Hinweise

- Der Befehl Where bezieht sich auf Felder, die nicht mit Hilfe einer Aggregatfunktion zusammengefasst werden. Die Where-Klausel filtert einzelne Datensätze.
- Der Befehl Having bezieht sich auf Felder, deren Felder mit Hilfe von Aggregatfunktionen berechnet wurden. Die Having-Klausel filtert Gruppen von Datensätze.

Kombination

```
SELECT
customer.last_Name,  SUM(amount) AS PaymentForCustomer
FROM customer
INNER JOIN payment ON customer.customer_id = payment.customer_id

WHERE customer.last_Name LIKE 'A%'

GROUP BY customer.last_Name

HAVING SUM(amount) > 100

ORDER BY customer.last_Name;
```