合肥大學
HEFEI UNIVERSITY

Datenbanken

8. Fabrik-Datenbank: Tabelle *product*

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-06-28

Databases

Dies ist ein Kurs über Datenbanken an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/databases (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielen finden Sie unter https://github.com/thomasWeise/databasesCode.

Outline

Einleitung

Einleitung

- Nun wollen wir die eigentliche Datenbank entwickeln.

Einleitung

- Nun wollen wir die eigentliche Datenbank entwickeln.
- Normalerweise würden wir dabei einem Designprozess folgen, wir würden die Anforderungen aufschreiben und analysieren, ein konzeptuelles Modell entwickeln und daraus ein logisches Modell ableiten...

Einleitung

- Nun wollen wir die eigentliche Datenbank entwickeln.
- Normalerweise würden wir dabei einem Designprozess folgen, wir würden die Anforderungen aufschreiben und analysieren, ein konzeptuelles Modell entwickeln und daraus ein logisches Modell ableiten...
- ...das machen wir jetzt nicht – unser Ziel ist *Learning by Doing*.

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].
- Auch dort sind Daten in Tabellen organisiert.

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können Text eingeben, der dem vorgegebenen Datentyp entspricht.

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können Text eingeben, der dem vorgegebenen Datentyp entspricht.
- Und es kann mehrere verschiedene Tabellen geben.

Wir nutzen eine Relationale Datenbank

- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel[1,8,14] oder LibreOffice Calc[5,6,11].
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können Text eingeben, der dem vorgegebenen Datentyp entspricht.
- Und es kann mehrere verschiedene Tabellen geben.
- Und die Datensätze in verschiedenen Tabellen können miteinander fest verbunden sein.

Struktur der Datenbank

- Dieses Format erlaubt es uns, unsere Daten nach Kategorie einzuteilen.

Struktur der Datenbank

- Dieses Format erlaubt es uns, unsere Daten nach Kategorie einzuteilen.
- Wir werden eine Tabelle für die *Producte* unseres Betriebs erstellen.

Struktur der Datenbank

- Dieses Format erlaubt es uns, unsere Daten nach Kategorie einzuteilen.
- Wir werden eine Tabelle für die *Producte* unseres Betriebs erstellen.
- Wir werden eine Tabelle für die *Kunden* unseres Betriebs erstellen.

Struktur der Datenbank

- Dieses Format erlaubt es uns, unsere Daten nach Kategorie einzuteilen.
- Wir werden eine Tabelle für die *Producte* unseres Betriebs erstellen.
- Wir werden eine Tabelle für die *Kunden* unseres Betriebs erstellen.
- Wir werden eine Tabelle für die *Bestellungen* der Kunden unseres Betriebs erstellen.

Struktur der Datenbank

- Dieses Format erlaubt es uns, unsere Daten nach Kategorie einzuteilen.
- Wir werden eine Tabelle für die *Producte* unseres Betriebs erstellen.
- Wir werden eine Tabelle für die *Kunden* unseres Betriebs erstellen.
- Wir werden eine Tabelle für die *Bestellungen* der Kunden unseres Betriebs erstellen.
- Fangen wir mit der Tabelle für die Produkte an.

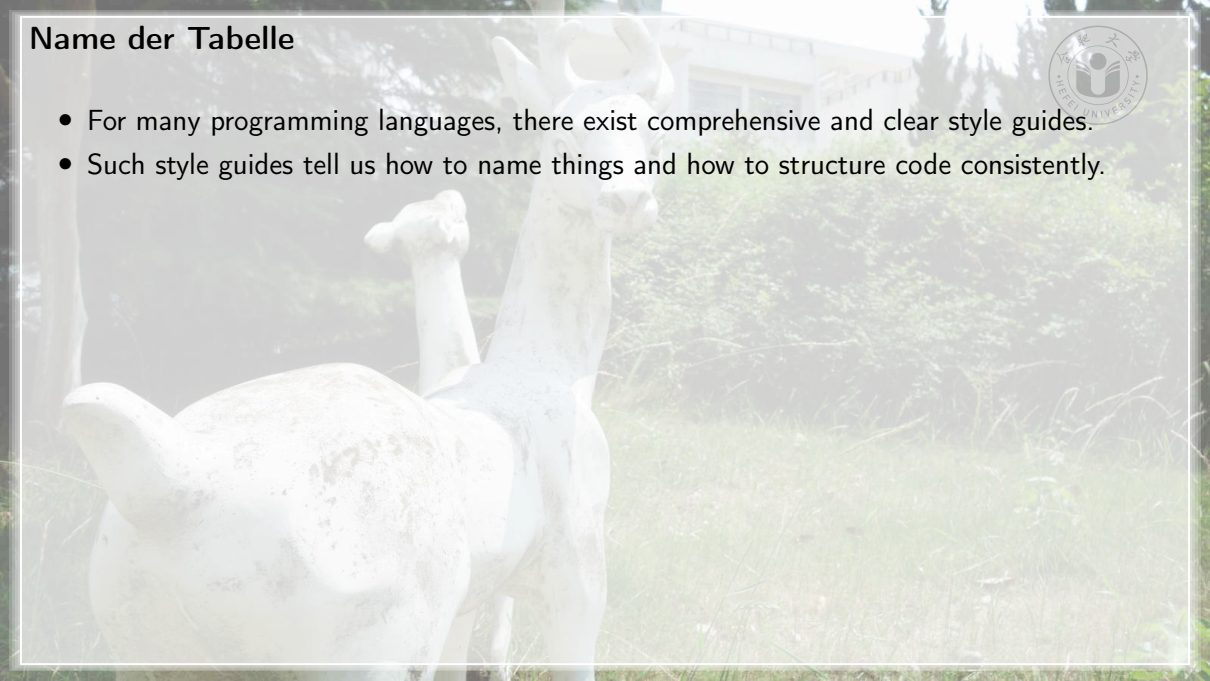
Erstellen der Tabelle

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.



Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.
- This is important, because we always work together with other people on projects and everybody should be able to read everyone's code.

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.
- This is important, because we always work together with other people on projects and everybody should be able to read everyone's code.
- For SQL, there does not exist one generally accepted best practice standard style guide.

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.
- This is important, because we always work together with other people on projects and everybody should be able to read everyone's code.
- For SQL, there does not exist one generally accepted best practice standard style guide.
- We will still try to define some general rules.

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.
- This is important, because we always work together with other people on projects and everybody should be able to read everyone's code.
- For SQL, there does not exist one generally accepted best practice standard style guide.
- We will still try to define some general rules.

Gute Praxis

Tabellennamen sollten (Englische) Nomen im Singular sein, mit Kleinbuchstaben geschrieben, und ohne Prefix (z.B. kein "tbl_" am Anfange haben)[3].

Name der Tabelle

- For many programming languages, there exist comprehensive and clear style guides.
- Such style guides tell us how to name things and how to structure code consistently.
- This is important, because we always work together with other people on projects and everybody should be able to read everyone's code.
- For SQL, there does not exist one generally accepted best practice standard style guide.
- We will still try to define some general rules.

Gute Praxis

Tabellennamen sollten (Englische) Nomen im Singular sein, mit Kleinbuchstaben geschrieben, und ohne Prefix (z.B. kein "tbl_" am Anfange haben)[3].

- Nennen wir unsere Tabelle für Produkte also `products`.

Grobstruktur

- Erstellen wir also die Tabelle `product`.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis. Der Preis ist eine Zahl die den Anforderungen an eine Geldmenge entsprechen muss.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis. Der Preis ist eine Zahl die den Anforderungen an eine Geldmenge entsprechen muss.
  - Um die Produkte auszuliefern, kommt jedes Produkt in ein Schachtel.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis. Der Preis ist eine Zahl die den Anforderungen an eine Geldmenge entsprechen muss.
  - Um die Produkte auszuliefern, kommt jedes Produkt in ein Schachtel. Diese Schachteln haben eine Breite, Höhe, und Tiefe, sowie (wenn das Produkt eingepackt ist) ein bestimmtes Gewicht.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis. Der Preis ist eine Zahl die den Anforderungen an eine Geldmenge entsprechen muss.
  - Um die Produkte auszuliefern, kommt jedes Produkt in ein Schachtel. Diese Schachteln haben eine Breite, Höhe, und Tiefe, sowie (wenn das Produkt eingepackt ist) ein bestimmtes Gewicht. Das sind also noch vier Zahlen, die wir speichern müssen.

Grobstruktur

- Erstellen wir also die Tabelle `product`.
- Welche Spalten brauchen wir?
  - Jedes Produkt hat einen *Namen*. Der Name ist im Grunde Text, also brauchen wir eine Spalte für Text.
  - Jedes Produkt hat einen Preis. Der Preis ist eine Zahl die den Anforderungen an eine Geldmenge entsprechen muss.
  - Um die Produkte auszuliefern, kommt jedes Produkt in ein Schachtel. Diese Schachteln haben eine Breite, Höhe, und Tiefe, sowie (wenn das Produkt eingepackt ist) ein bestimmtes Gewicht. Das sind also noch vier Zahlen, die wir speichern müssen.
- Das ist also die grundlegende Struktur unserer Tabelle.

Grobstruktur

- Das ist also die grundlegende Struktur unserer Tabelle.

id	name	price	weight	width	height	depth
1	Shoe, Size 36	150.99	1300	350	250	130
2	Shoe, Size 37	152.99	1325	350	250	130
3	Shoe, Size 38	154.99	1350	350	250	130
4	Shoe, Size 39	156.99	1375	350	250	130
5	Shoe, Size 40	158.99	1400	350	250	130
6	Shoe, Size 41	160.99	1425	350	250	130
7	Shoe, Size 42	162.99	1450	350	250	130
8	Shoe, Size 43	164.99	1475	350	250	130
9	Small Purse	100	500	350	250	130
10	Medium Purse	120	750	400	300	200
11	Large Purse	150	1500	600	300	250
…	…	…	…	…	…	…
		RMB	g	mm	mm	mm

SQL

- Diese Struktur ist im Grunde recht einfach.

SQL

- Diese Struktur ist im Grunde recht einfach.
- Nun müssen wir sie in SQL-Kommandos übersetzen.

SQL

- Diese Struktur ist im Grunde recht einfach.
- Nun müssen wir sie in SQL-Kommandos übersetzen.
- Wir müssen dazu Namen, Datentypen, und Einschränkungen für die Spalten festlegen.

SQL

- Diese Struktur ist im Grunde recht einfach.
- Nun müssen wir sie in SQL-Kommandos übersetzen.
- Wir müssen dazu Namen, Datentypen, und Einschränkungen für die Spalten festlegen.
- Dann können wir die SQL-Kommandos an das PostgreSQL DBMS schicken und die Tabelle erstellen.

SQL

- Diese Struktur ist im Grunde recht einfach.
- Nun müssen wir sie in SQL-Kommandos übersetzen.
- Wir müssen dazu Namen, Datentypen, und Einschränkungen für die Spalten festlegen.
- Dann können wir die SQL-Kommandos an das PostgreSQL DBMS schicken und die Tabelle erstellen.
- Dies kann dann innerhalb einer interaktiven psql-Session oder mit Hilfe eines Skripts erfolgen.

SQL

- Diese Struktur ist im Grunde recht einfach.
- Nun müssen wir sie in SQL-Kommandos übersetzen.
- Wir müssen dazu Namen, Datentypen, und Einschränkungen für die Spalten festlegen.
- Dann können wir die SQL-Kommandos an das PostgreSQL DBMS schicken und die Tabelle erstellen.
- Dies kann dann innerhalb einer interaktiven psql-Session oder mit Hilfe eines Skripts erfolgen.
- Schauen wir uns das mal an.

Interaktive PSQL Session – Teil 1

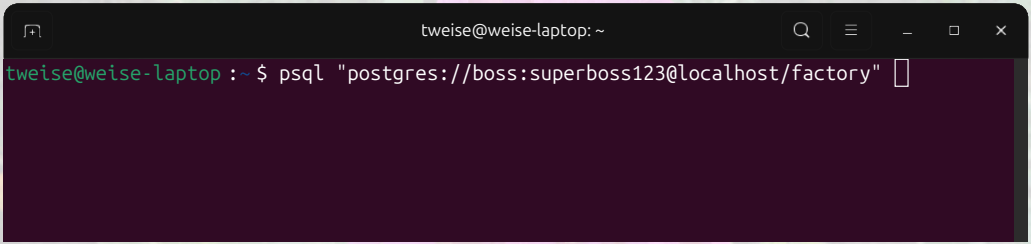
- Wir erstellen das Kommando zuerst insgesamt und erklären dann Schritt-für-Schritt seine Komponenten.

Interaktive PSQL Session – Teil 1

- Wir erstellen das Kommando zuerst insgesamt und erklären dann Schritt-für-Schritt seine Komponenten.
- Wir loggen uns jetzt in unseren PostgreSQL-Server ein und erstellen die neue Tabelle.

Interaktive PSQL Session – Teil 1

- Wir starten psql mit der passenden Verbindungs-URI. Diesmal verwenden with den Benutzer `boss` mit Passwort `superboss123` und arbeiten auf der Datenbank `factory`.



Interaktive PSQL Session – Teil 1

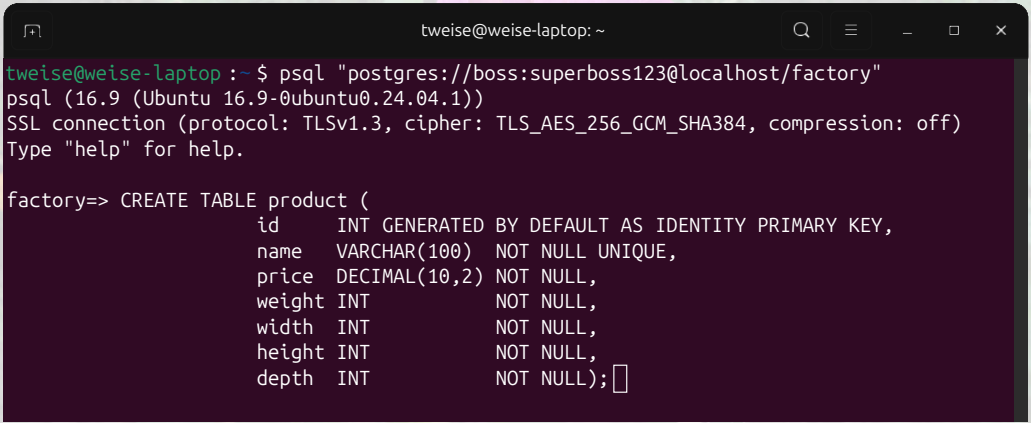
- psql läuft und ist mit dem PostgreSQL DBMS verbunden.

```
tweise@weise-laptop: ~
tweise@weise-laptop :~ $ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=>
```

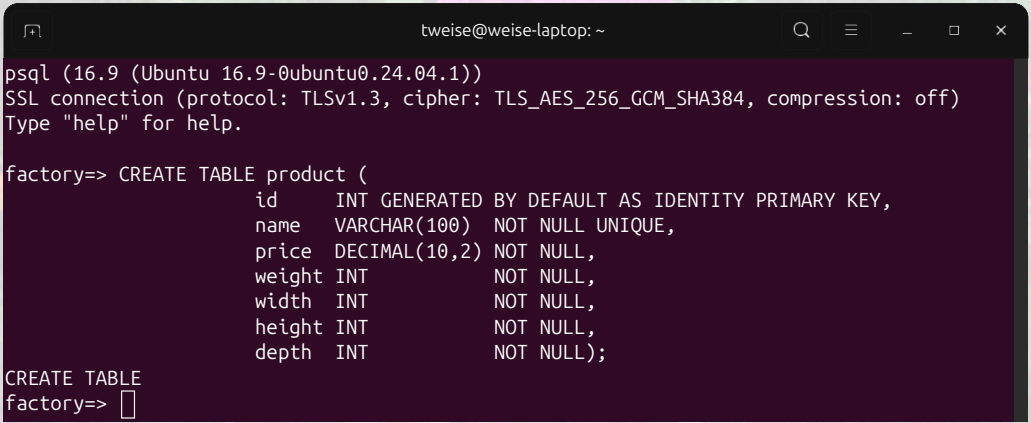
Interaktive PSQL Session – Teil 1

- Mit diesem `CREATE TABLE` Kommando – das wir gleich im Detail anschauen – wird die Tabelle mit allen notwendigen Spalten erstellt.



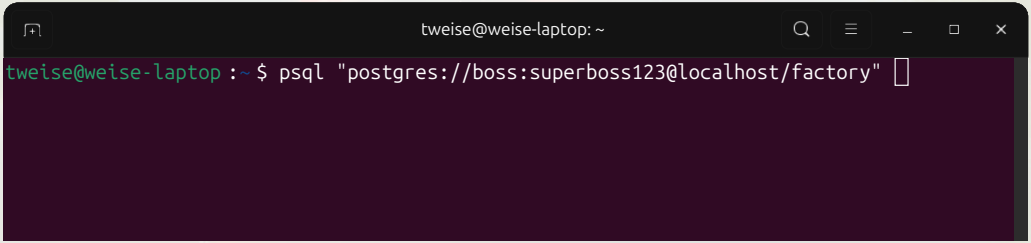
Interaktive PSQL Session – Teil 1

- Das Kommando wird erfolgreich ausgeführt und `CREATE TABLE` wird uns als Antwort ausgegeben.



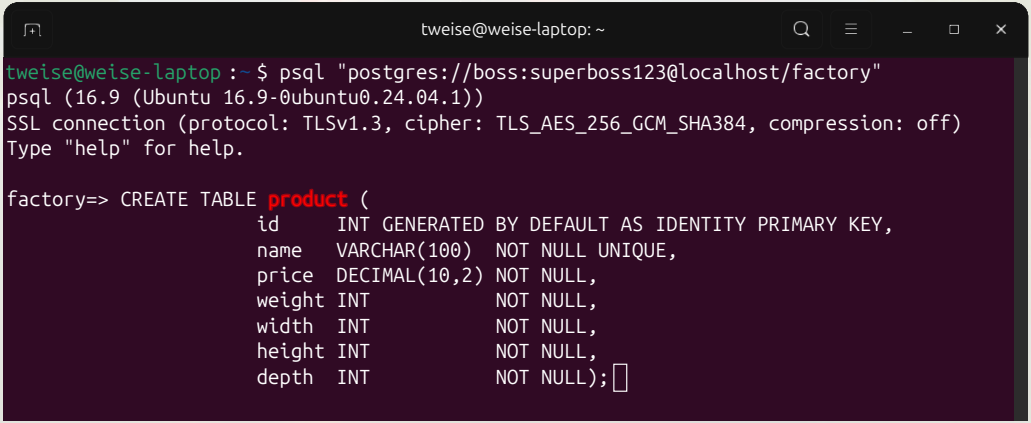
Command Structure

- `CREATE TABLE` ist das SQL-Kommando um eine Tabelle anzulegen.



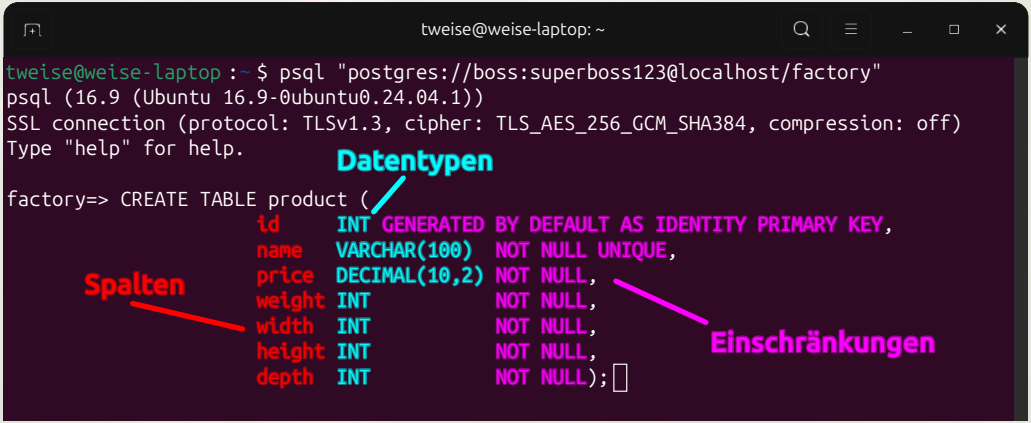
Command Structure

- `product` ist der Name der Tabelle, die wir anlegen wollen.



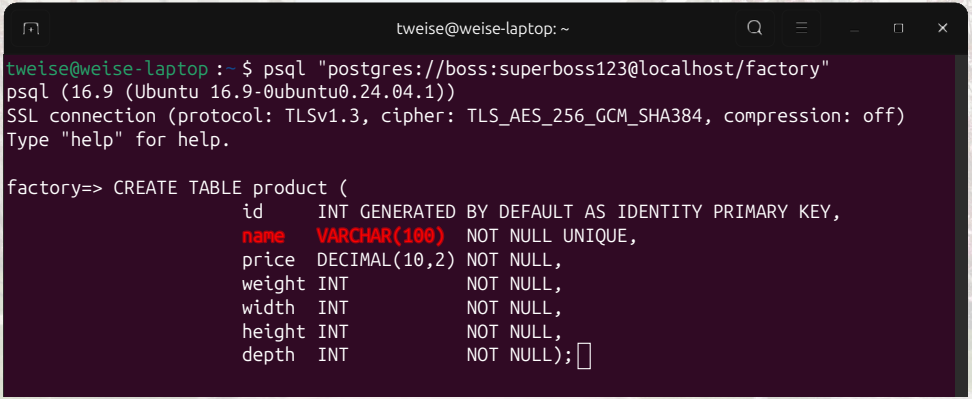
Command Structure

- Dann folgen die Namen, Datentypen, und Einschränkungen der Spalten unserer Tabelle. Diese sind von Klammern umrahmt (`(...)`) und durch Kommas (`,`) getrennt.



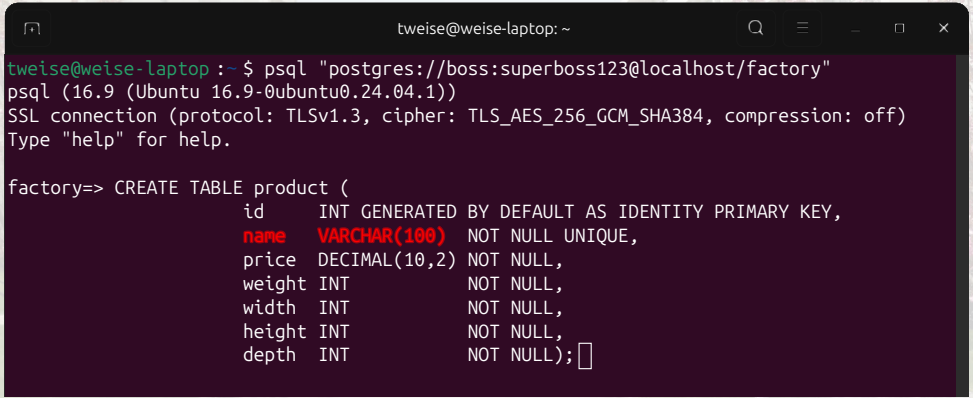
Die Spalte `name`

- Wir legen eine Spalte `name` für die Namen der Produkte an.



Die Spalte `name`

- Wir legen eine Spalte `name` für die Namen der Produkte an. Die Namen sind Text, haben aber keine feste Lände.



Die Spalte `name`

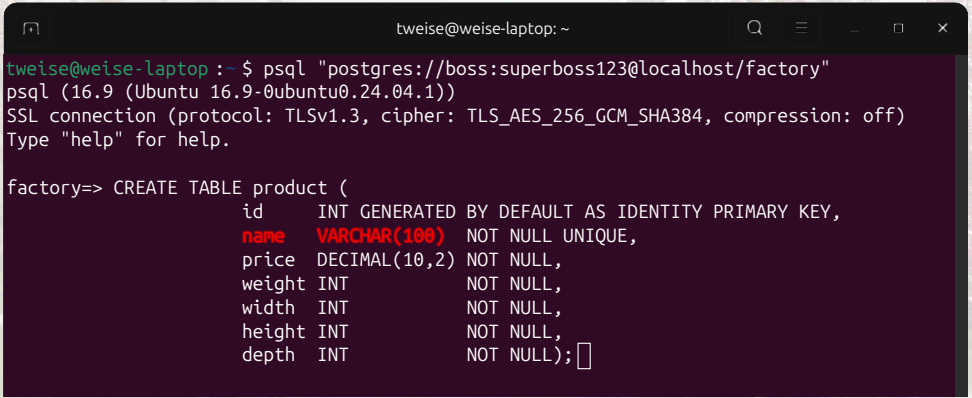
- Wir legen eine Spalte `name` für die Namen der Produkte an. Die Namen sind Text, haben aber keine feste Länge. Dafür gibt es den Datentyp `VARCHAR`.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                    id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                    name   VARCHAR(100)  NOT NULL UNIQUE,
                    price  DECIMAL(10,2) NOT NULL,
                    weight INT           NOT NULL,
                    width  INT           NOT NULL,
                    height INT           NOT NULL,
                    depth  INT           NOT NULL);
```

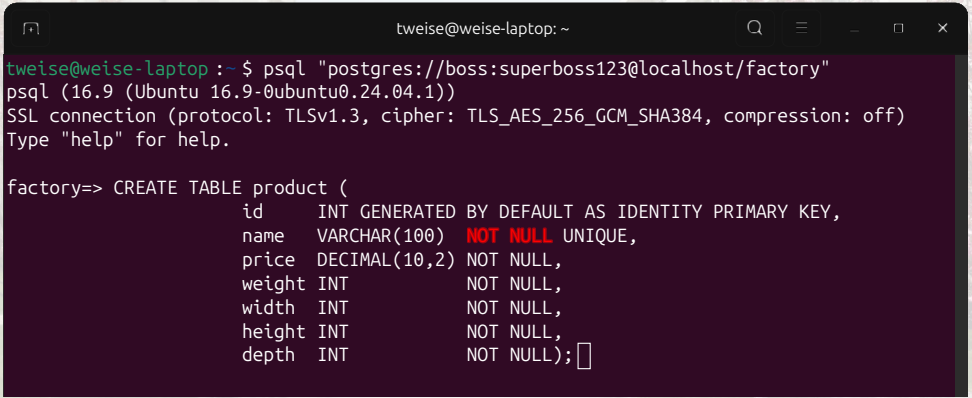
Die Spalte `name`

- Wir legen eine Spalte `name` für die Namen der Produkte an. Die Namen sind Text, haben aber keine feste Länge. Dafür gibt es den Datentyp `VARCHAR`. In Klammern geben wir eine Maximallänge an. Hier ist 100 Zeichen eine vernünftige Wahl.



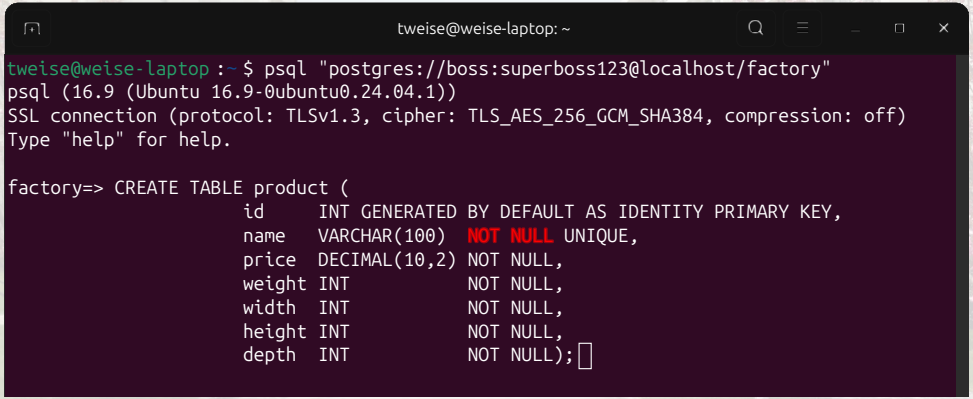
Die Spalte `name`

- Jedes Produkt muss einen Namen haben.



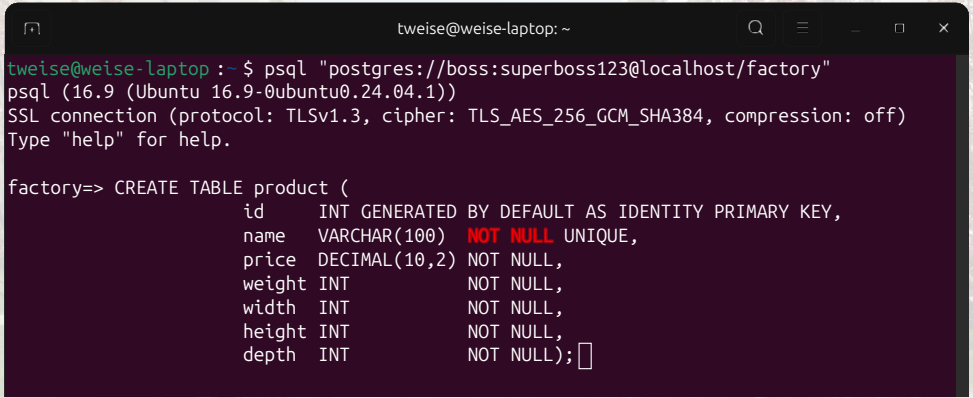
Die Spalte `name`

- Jedes Produkt muss einen Namen haben. Es kann kein Produkt ohne Name geben.



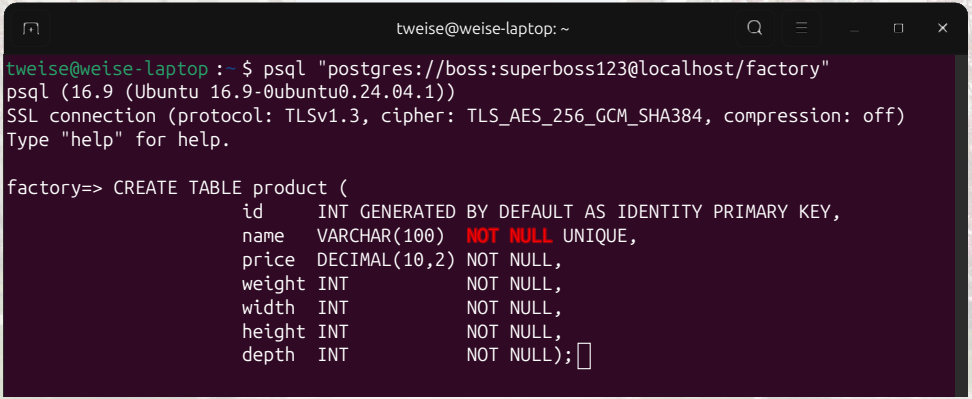
Die Spalte `name`

- Jedes Produkt muss einen Namen haben. Es kann kein Produkt ohne Name geben. Wir schreiben daher `NOT NULL` hinter den Datentypen.



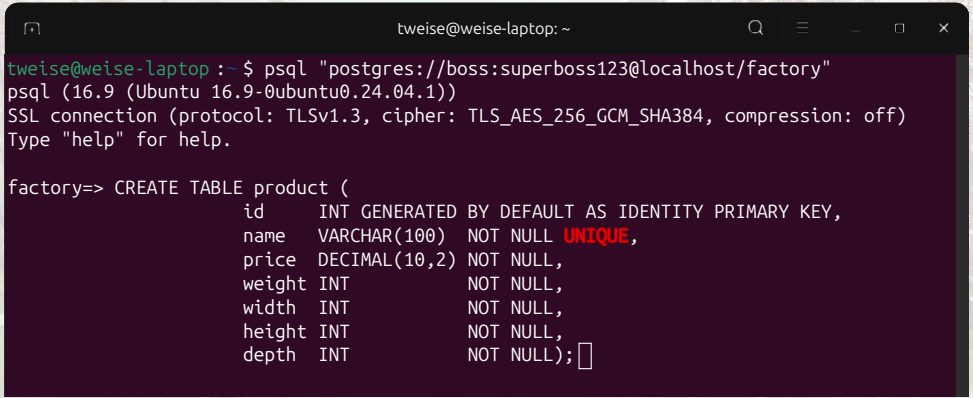
Die Spalte `name`

- Jedes Produkt muss einen Namen haben. Es kann kein Produkt ohne Name geben. Wir schreiben daher `NOT NULL` hinter den Datentypen. Diese Einschränkung stellt sicher, dass jeder Datensatz immer einen Wert in der Spalte `name` haben muss.



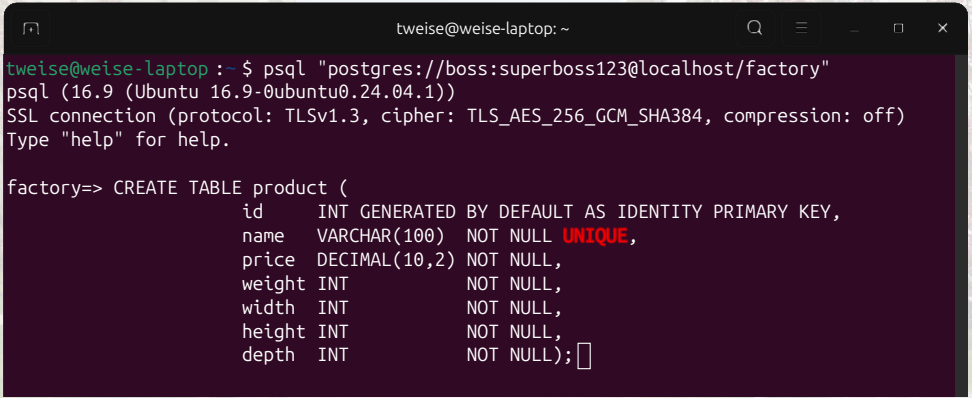
Die Spalte `name`

- Die Produktnamen müssen eindeutig sein.



Die Spalte `name`

- Die Produktnamen müssen eindeutig sein. Zwei verschiedene Produkte dürfen niemals den gleichen Namen haben.



Die Spalte `name`

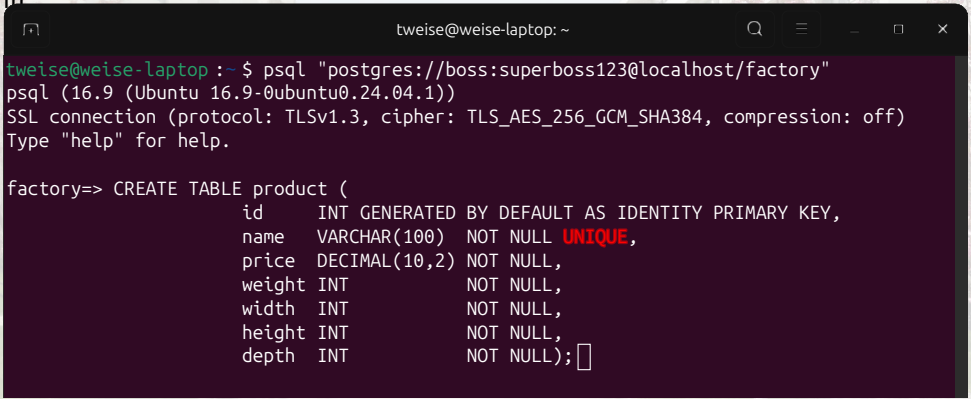
- Die Produktnamen müssen eindeutig sein. Zwei verschiedene Produkte dürfen niemals den gleichen Namen haben. Wir schreiben daher `UNIQUE` hinter den Datentypen.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                    id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                    name   VARCHAR(100)  NOT NULL UNIQUE,
                    price  DECIMAL(10,2) NOT NULL,
                    weight INT           NOT NULL,
                    width  INT           NOT NULL,
                    height INT           NOT NULL,
                    depth  INT           NOT NULL);
```

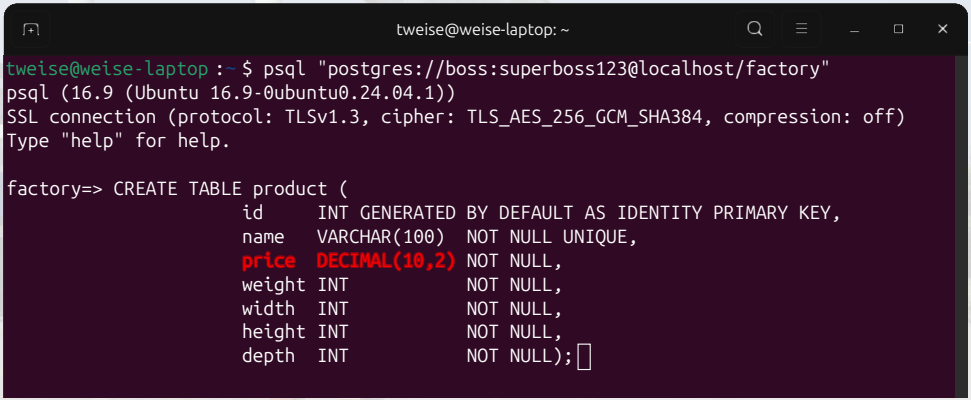
Die Spalte `name`

- Die Produktnamen müssen eindeutig sein. Zwei verschiedene Produkte dürfen niemals den gleichen Namen haben. Wir schreiben daher `UNIQUE` hinter den Datentypen. Diese Einschränkung stellt sicher, dass jeder Wert in der Spalte `name` nur einmal vorkommen kann.



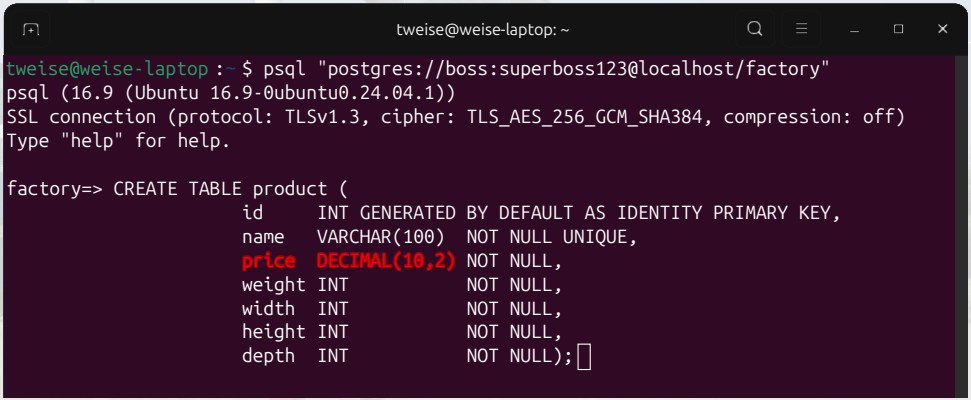
Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an.



Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.



Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie “1,34€” oder “12.99元” oder “$3.99” sehen auf den ersten Blick wie Fließkommazahlen aus.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
```

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus.
    - Sie kennen Fließkommazahlen aus Python (Datentype `float`) oder Java und C (Datentypen `float`, `double`, ...).

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus.
    - Sie kennen Fließkommazahlen aus Python (Datentype `float`) oder Java und C (Datentypen `float`, `double`, ...).
    - Aber Fließkommazahlen sind ungenau!

Gute Praxis

Nehmen Sie immer an, dass `float`-Werte ungenau sind. Nehmen Sie NIEMALS an, dass diese exact sind[2,9,12].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus.
    - Sie kennen Fließkommazahlen aus Python (Datentype `float`) oder Java und C (Datentypen `float`, `double`, ...).
    - Aber Fließkommazahlen sind ungenau!
    - Beispiel: `0.1 + 0.1 + 0.1 - 0.3` ergiebt `5.551115123125783e-17` in Python...

Gute Praxis

Nehmen Sie immer an, dass `float`-Werte ungenau sind. Nehmen Sie NIEMALS an, dass diese exact sind[2,9,12].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus.
    - Sie kennen Fließkommazahlen aus Python (Datentype `float`) oder Java und C (Datentypen `float`, `double`, ...).
    - Aber Fließkommazahlen sind ungenau!
    - Beispiel: `0.1 + 0.1 + 0.1 - 0.3` ergiebt `5.551115123125783e-17` in Python...

Gute Praxis

Repräsentieren Sie niemals Geldwerte mit Fließkommazahlen[10,13].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie “1,34€” oder “12.99元” oder “$3.99” sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!

Gute Praxis

Repräsentieren Sie niemals Geldwerte mit Fließkommazahlen[10,13].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie “1,34€” oder “12.99元” oder “$3.99” sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分.

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分.
    - Dann muss man sie aber jedesmal in "umrechnen", wenn man sie anzeigen oder mit ihnen rechnen will.

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分.
    - Dann muss man sie aber jedesmal in "umrechnen", wenn man sie anzeigen oder mit ihnen rechnen will.
    - Aber das ist fehleranfällig[13].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie “1,34€” oder “12.99元” oder “$3.99” sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分. Preise sind aber keine Ganzzahlen.

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie “1,34€” oder “12.99元” oder “$3.99” sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分. Preise sind aber keine Ganzzahlen.
  - Der Datentype `DECIMAL(x, y)` erlaubt es uns, Zahlen mit `x` Stellen (`y` davon nach dem Komma) exakt darzustellen.

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分. Preise sind aber keine Ganzzahlen.
  - Der Datentype `DECIMAL(x, y)` erlaubt es uns, Zahlen mit `x` Stellen (`y` davon nach dem Komma) exakt darzustellen.

Gute Praxis

Geldwerte müssen mit dem Datentype `DECIMAL` gespeicher werden[4,13].

Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer.
  - Preise wie "1,34€" oder "12.99元" oder "$3.99" sehen auf den ersten Blick wie Fließkommazahlen aus. Preise sind aber keine Fließkommazahlen!
  - Preise könnten Ganzzahlen sein, z.B. die Anzahl Cent oder 分. Preise sind aber keine Ganzzahlen.
  - Der Datentype `DECIMAL(x, y)` erlaubt es uns, Zahlen mit `x` Stellen (`y` davon nach dem Komma) exakt darzustellen.
    - Mit `price DECIMAL(10, 2)` können wir Zahlen mit 10 Ziffern, 2 davon nach dem Komma, exakt speichern[7].

Gute Praxis

Geldwerte müssen mit dem Datentype `DECIMAL` gespeicher werden[4,13].

Die Spalte `price`

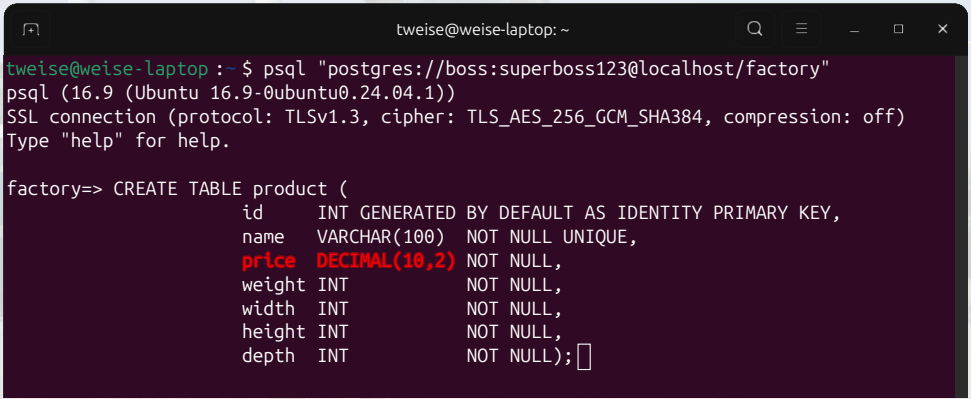
- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer. Wir benutzen den Datentyp `DECIMAL(10, 2)`.

Gute Praxis

Geldwerte müssen mit dem Datentype `DECIMAL` gespeicher werden[4,13].

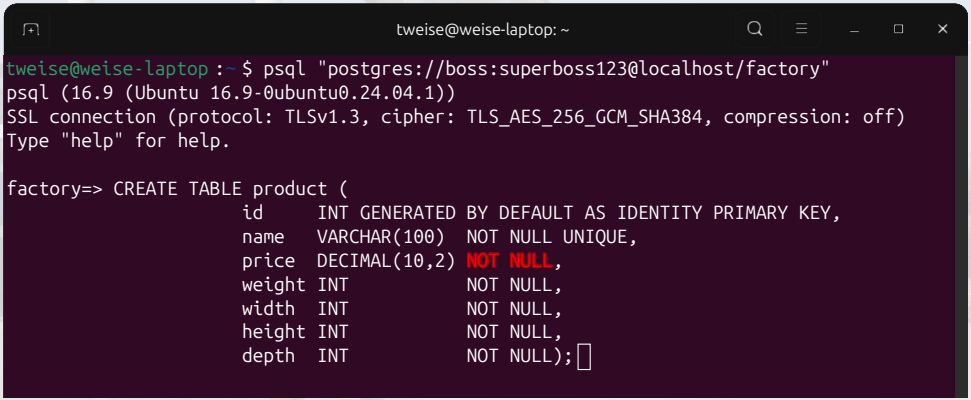
Die Spalte `price`

- Wir legen eine Spalte `price` für den Preis der Produkte an. Die Auswahl des richtigen Datentypen ist diesmal schwer. Wir benutzen den Datentyp `DECIMAL(10, 2)`.



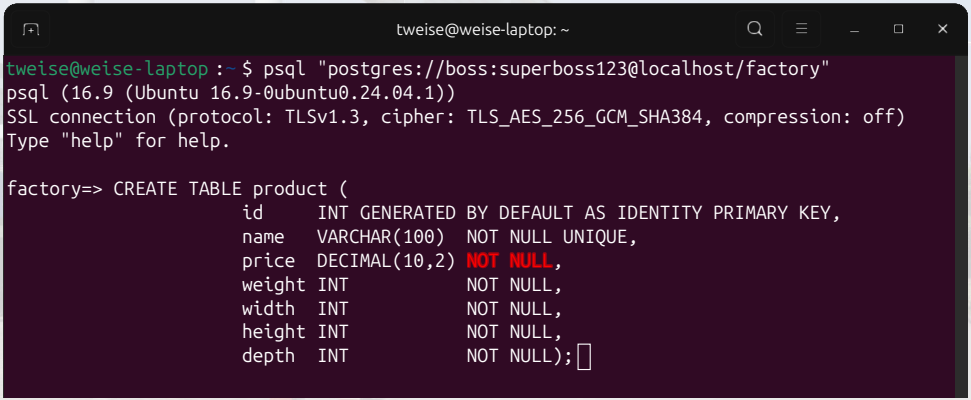
Die Spalte `price`

- Jedes Produkt muss einen Preis haben.



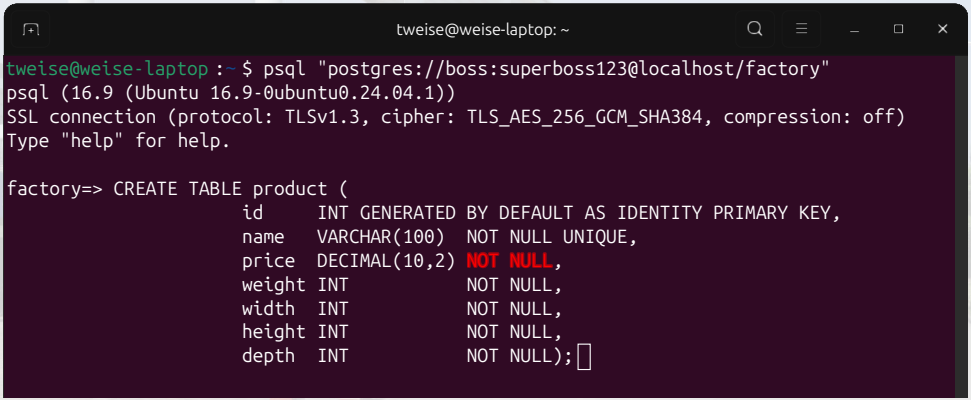
Die Spalte `price`

- Jedes Produkt muss einen Preis haben. Es kann kein Produkt ohne Preis geben.



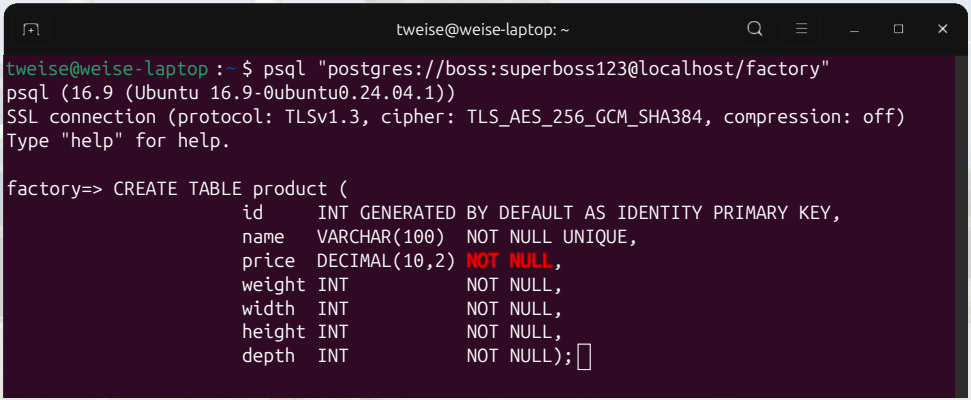
Die Spalte `price`

- Jedes Produkt muss einen Preis haben. Es kann kein Produkt ohne Preis geben. Wir schreiben daher `NOT NULL` hinter den Datentypen.



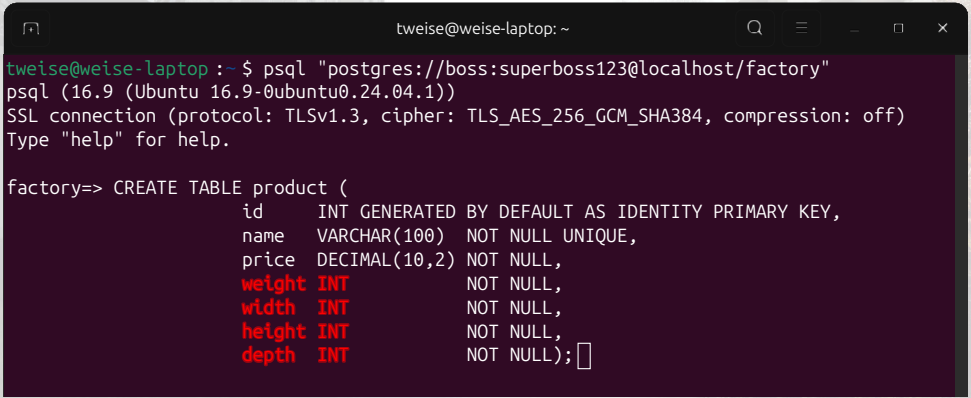
Die Spalte `price`

- Jedes Produkt muss einen Preis haben. Es kann kein Produkt ohne Preis geben. Wir schreiben daher `NOT NULL` hinter den Datentypen. Diese Einschränkung stellt sicher, dass jeder Datensatz immer einen Wert in der Spalte `preis` haben muss.



Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.



Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.
  - Das Gewicht soll in Gram in der Spalte `weight` gespeichert werden.

Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.
  - Das Gewicht soll in Gram in der Spalte `weight` gespeichert werden.
  - Die Breite soll in Millimeter in der Spalte `width` gespeichert werden.

Die Spalten `weight`, `width`, `height`, und `depth`

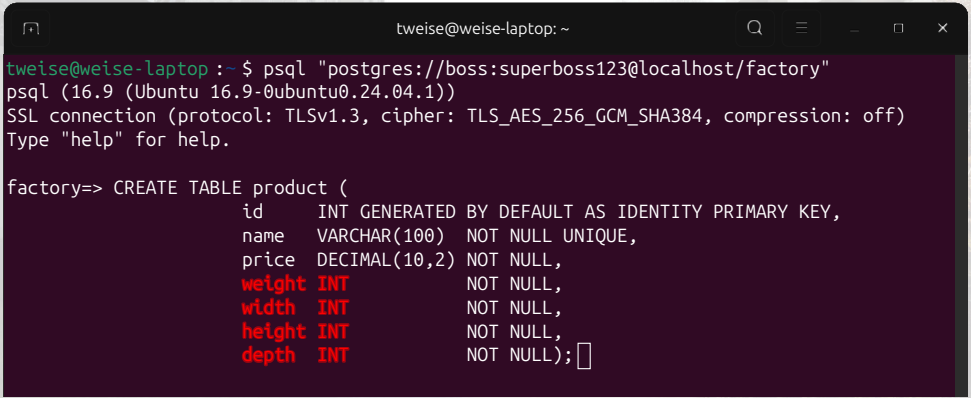
- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.
  - Das Gewicht soll in Gram in der Spalte `weight` gespeichert werden.
  - Die Breite soll in Millimeter in der Spalte `width` gespeichert werden.
  - Die Höhe soll in Millimeter in der Spalte `height` gespeichert werden.

Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.
  - Das Gewicht soll in Gram in der Spalte `weight` gespeichert werden.
  - Die Breite soll in Millimeter in der Spalte `width` gespeichert werden.
  - Die Höhe soll in Millimeter in der Spalte `height` gespeichert werden.
  - Die Tiefe soll in Millimeter in der Spalte `depth` gespeichert werden.

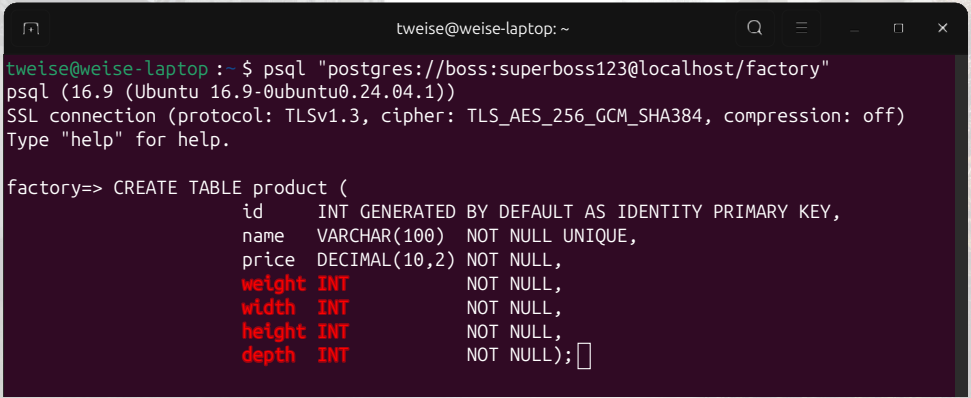
Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.



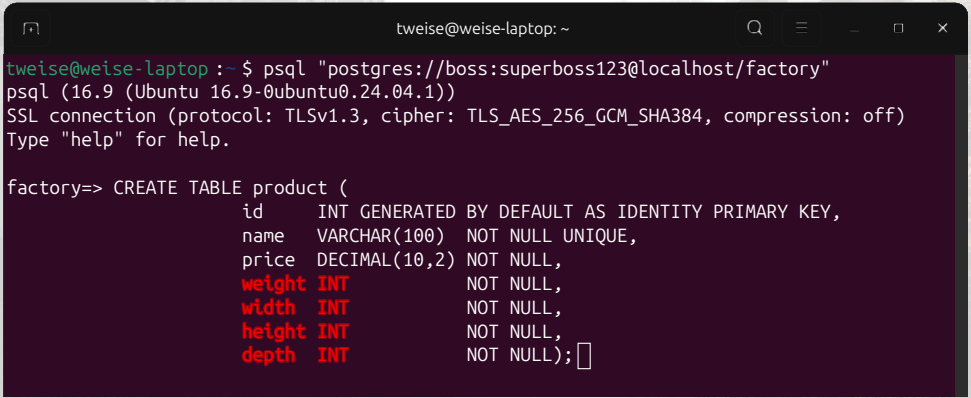
Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.



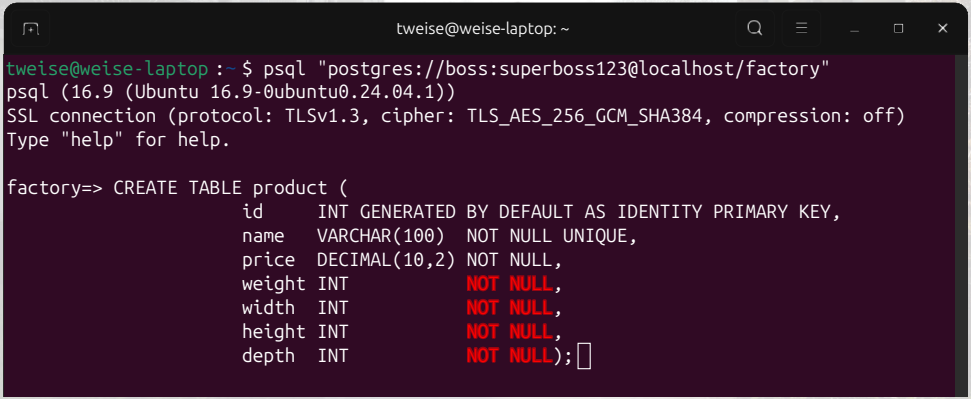
Die Spalten `weight`, `width`, `height`, und `depth`

- Wir legen die Spalten für das Gewicht und die Dimensionen der Boxen für unsere Produkte an.



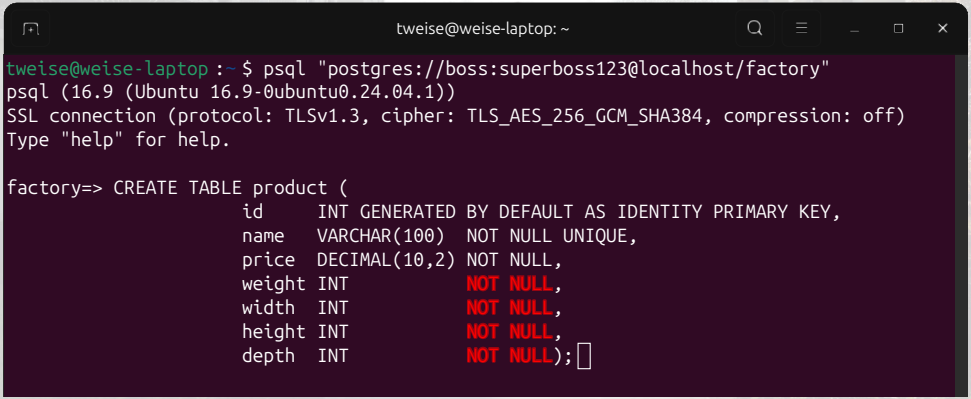
Die Spalten `weight`, `width`, `height`, und `depth`

- Jedes Produkt muss ein Gewicht und alle Box-Dimensionen haben.



Die Spalten `weight`, `width`, `height`, **und** `depth`

- Jedes Produkt muss ein Gewicht und alle Box-Dimensionen haben. Es kann kein Produkt ohne Gewicht oder mit unbekannter Größe geben.



Die Spalten `weight`, `width`, `height`, und `depth`

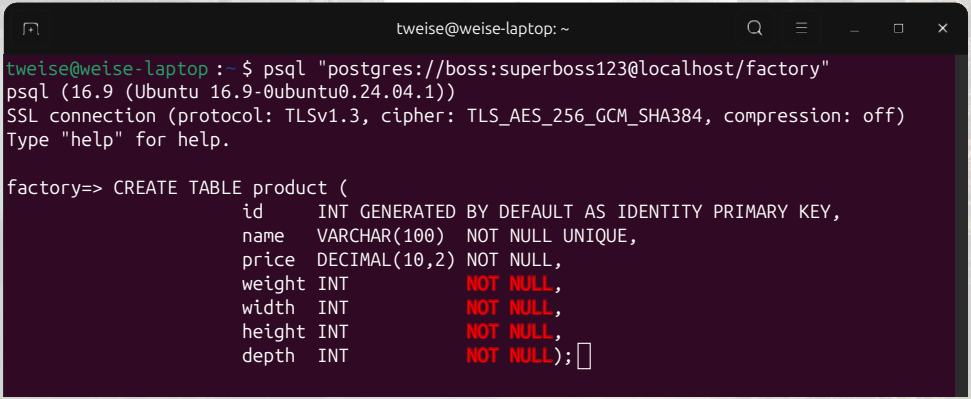
- Jedes Produkt muss ein Gewicht und alle Box-Dimensionen haben. Es kann kein Produkt ohne Gewicht oder mit unbekannter Größe geben. Wir schreiben daher `NOT NULL` hinter die Datentypen.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
```

Die Spalten `weight`, `width`, `height`, und `depth`

- Jedes Produkt muss ein Gewicht und alle Box-Dimensionen haben. Es kann kein Produkt ohne Gewicht oder mit unbekannter Größe geben. Wir schreiben daher `NOT NULL` hinter die Datentypen. Diese Einschränkung stellt sicher, dass jeder Datensatz immer Werte für alle vier Attribute haben muss.



Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),
  - die Tabelle für Produkte (die wir gerade erstellen),

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),
  - die Tabelle für Produkte (die wir gerade erstellen),
  - und eine Tabelle, die eine Zeile aus der Kundentabelle mit einer Zeile aus der Produkttabelle zu einer Bestellung “verbindet”.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),
  - die Tabelle für Produkte (die wir gerade erstellen),
  - und eine Tabelle, die eine Zeile aus der Kundentabelle mit einer Zeile aus der Produkttabelle zu einer Bestellung "verbindet".
- Damit das geht, müssen wir die Zeilen in der Kunden- und der Produkttabelle finden können.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),
  - die Tabelle für Produkte (die wir gerade erstellen),
  - und eine Tabelle, die eine Zeile aus der Kundentabelle mit einer Zeile aus der Produkttabelle zu einer Bestellung "verbindet".
- Damit das geht, müssen wir die Zeilen in der Kunden- und der Produkttabelle finden können.
- Jede Zeile muss eindeutig identifiziert werden können.

Primärschlüssel

- Die Tabellen in einer Datenbank existieren nicht alleine oder losgelöst.
- Sie referenzieren einander.
- Später wollen wir speichern, welcher Kunde welches Produkt bestellt hat.
- Dazu brauchen wir mindestens drei Tabellen:
  - eine Tabelle für Kunden (die wir noch nicht haben),
  - die Tabelle für Produkte (die wir gerade erstellen),
  - und eine Tabelle, die eine Zeile aus der Kundentabelle mit einer Zeile aus der Produkttabelle zu einer Bestellung "verbindet".
- Damit das geht, müssen wir die Zeilen in der Kunden- und der Produkttabelle finden können.
- Jede Zeile muss eindeutig identifiziert werden können.
- Dafür gibt es *Primärschlüssel*.

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt !

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).
- Wir könnten diese Spalte als Primärschlüssel nehmen.

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).
- Wir könnten diese Spalte als Primärschlüssel nehmen.
- Dann würde die Tabelle für Bestellungen für jede Bestellung den Namen des Produktes mitspeichern.

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).
- Wir könnten diese Spalte als Primärschlüssel nehmen.
- Dann würde die Tabelle für Bestellungen für jede Bestellung den Namen des Produktes mitspeichern.
- Über den Namen können wir dann die Eigenschaften, z.B. den Preis, auslesen.

Primärschlüssel

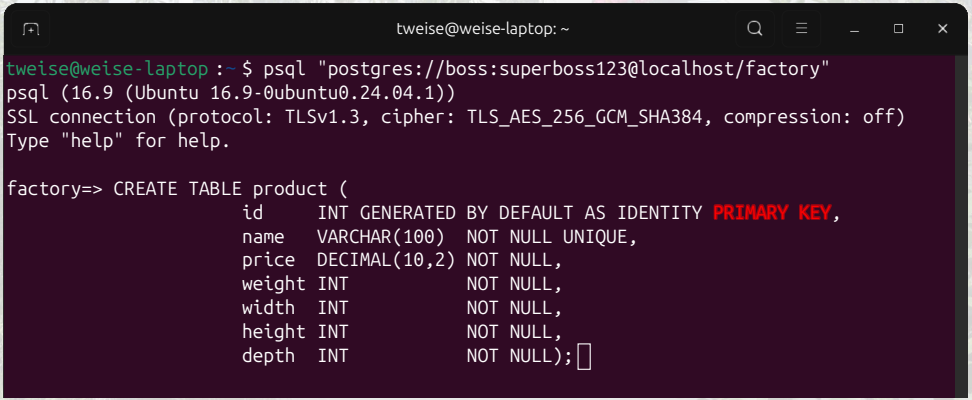
- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).
- Wir könnten diese Spalte als Primärschlüssel nehmen.
- Dann würde die Tabelle für Bestellungen für jede Bestellung den Namen des Produktes mitspeichern.
- Über den Namen können wir dann die Eigenschaften, z.B. den Preis, auslesen.
- Aber was, wenn wir irgendwann den Namen eines Produktes ändern?

Primärschlüssel

- Primärschlüssel sind (eine oder mehrere) Spalten einer Tabelle deren Werte die Zeilen eindeutig identifizieren.
- Das heißt, dass es keine zwei Zeilen einer Tabelle mit den selben Primärschlüsselwerten geben kann.
- Wir haben bereits so eine Spalte erstellt: `name`!
- Die Produktnamen, gespeichert in `name`, sind einmalig (`UNIQUE`).
- Wir könnten diese Spalte als Primärschlüssel nehmen.
- Dann würde die Tabelle für Bestellungen für jede Bestellung den Namen des Produktes mitspeichern.
- Über den Namen können wir dann die Eigenschaften, z.B. den Preis, auslesen.
- Aber was, wenn wir irgendwann den Namen eines Produktes ändern?
- Dann geht die Verbindung verloren...

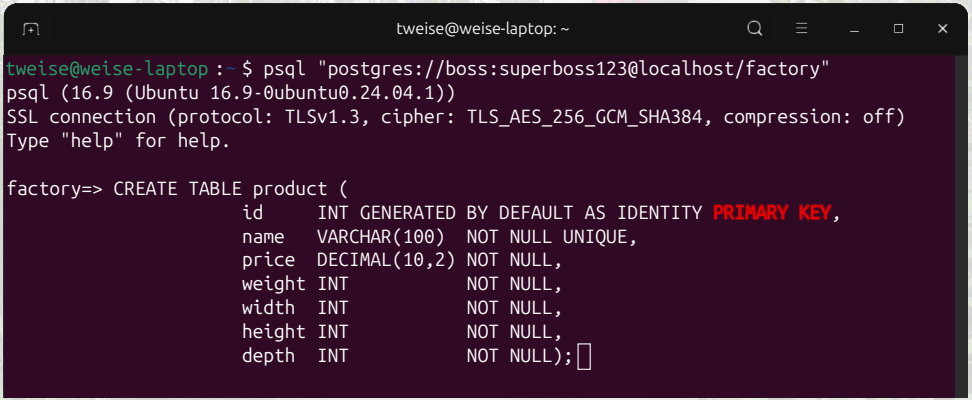
Die Spalte `id`

- Wir legen eine zusätzliche Spalte `id` als Primärschlüssel an.



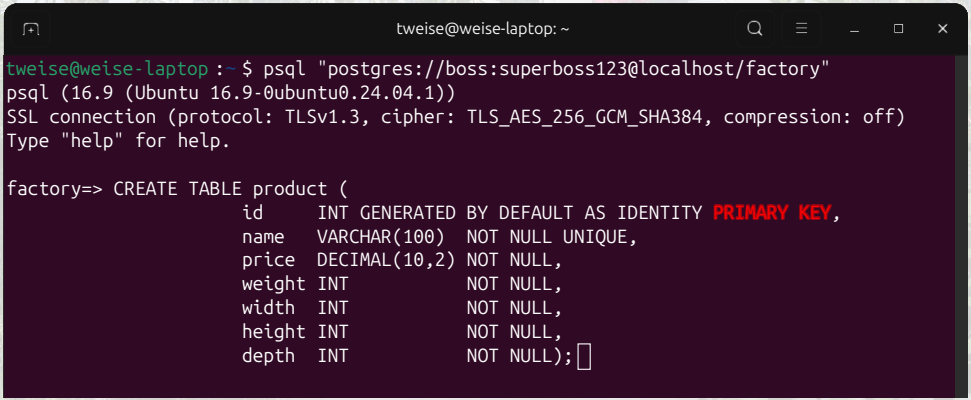
Die Spalte `id`

- Wir legen eine zusätzliche Spalte `id` als Primärschlüssel an. Wir markieren die Spalte as `PRIMARY KEY`.



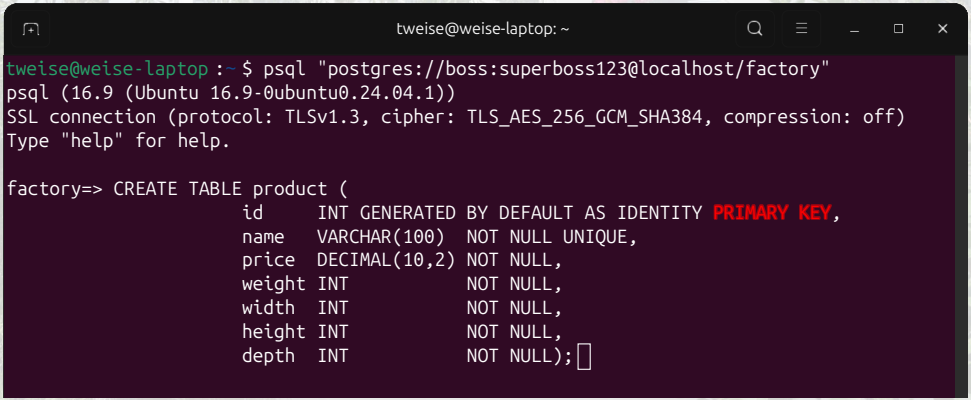
Die Spalte `id`

- Wir legen eine zusätzliche Spalte `id` als Primärschlüssel an. Wir markieren die Spalte as `PRIMARY KEY`. Dies impliziert `NOT NULL` und `UNIQUE`.



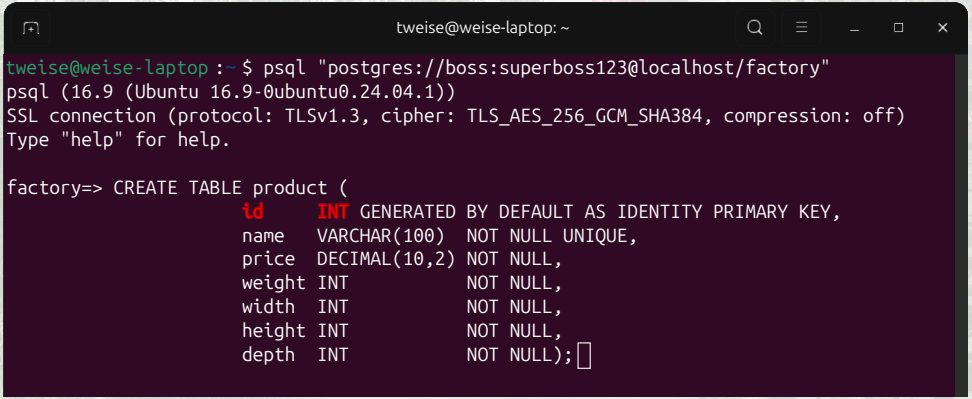
Die Spalte `id`

- Wir legen eine zusätzliche Spalte `id` als Primärschlüssel an. Wir markieren die Spalte as `PRIMARY KEY`. Dies impliziert `NOT NULL` und `UNIQUE`. Aber was kommt in diese Spalte?



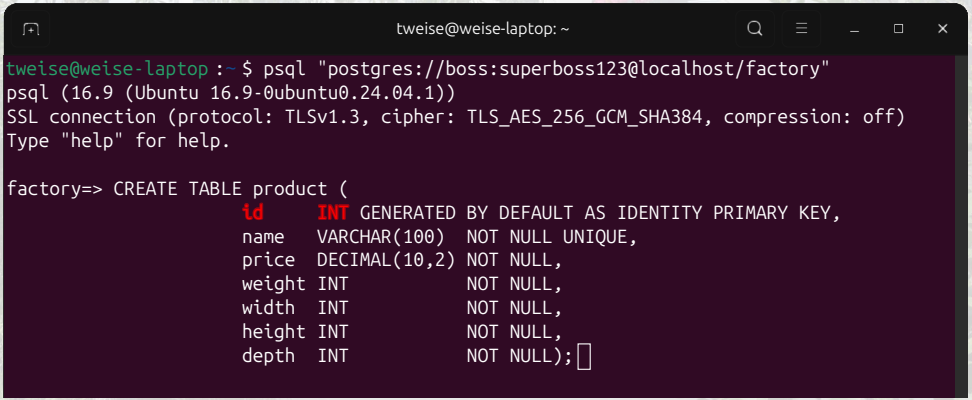
Die Spalte `id`

- Geben wir ihr ebenfalls den Datentyp `INT`.



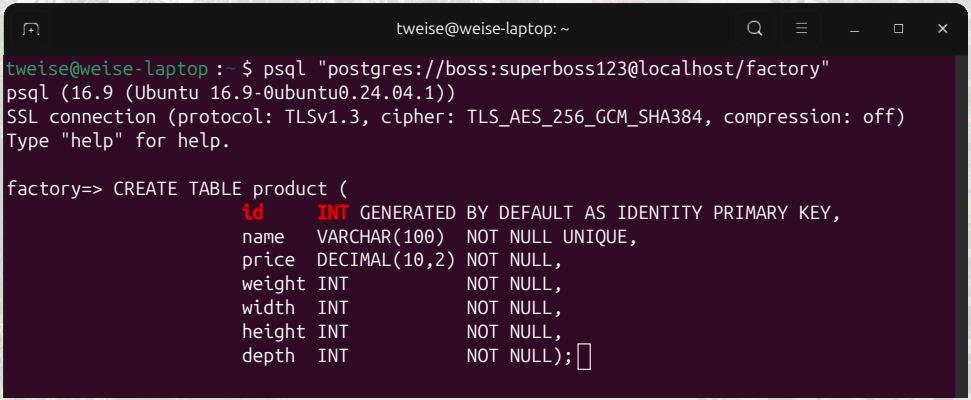
Die Spalte `id`

- Geben wir ihr ebenfalls den Datentyp `INT`. Die Primärschlüssel sollen also ganzzahlige Werte sein.



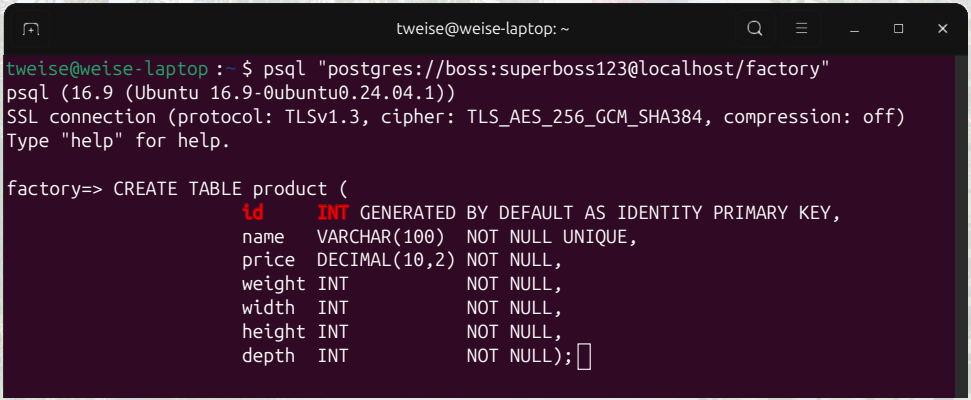
Die Spalte `id`

- Geben wir ihr ebenfalls den Datentyp `INT`. Die Primärschlüssel sollen also ganzzahlige Werte sein. Anders als die Produktnamen, werden wir diese niemals ändern.



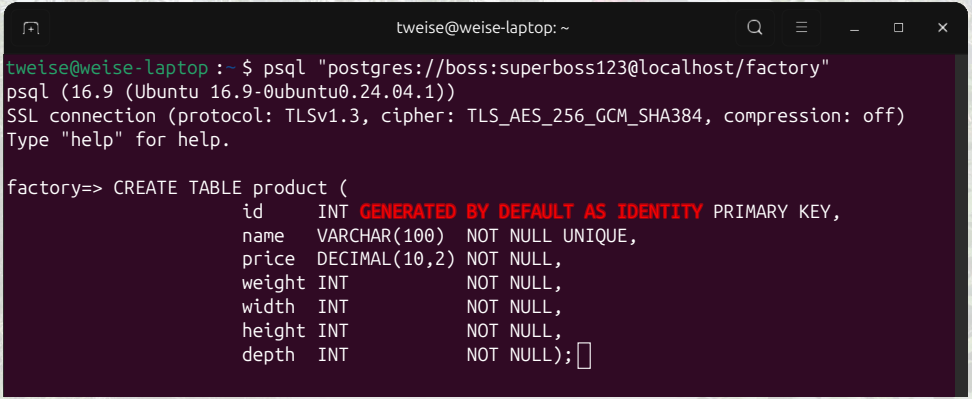
Die Spalte `id`

- Geben wir ihr ebenfalls den Datentyp `INT`. Die Primärschlüssel sollen also ganzzahlige Werte sein. Anders als die Produktnamen, werden wir diese niemals ändern. Aber was kommt in diese Spalte?



Die Spalte `id`

- Wir lassen sie automatisch von der Datenbank generieren!



Die Spalte `id`

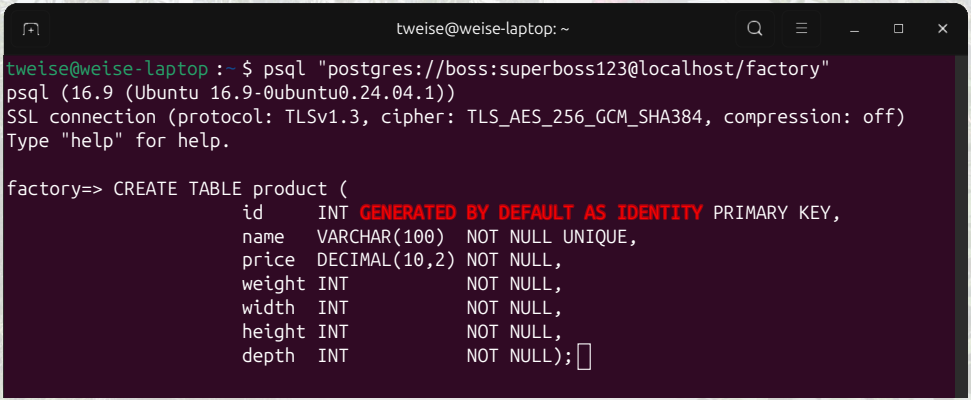
- Wir lassen sie automatisch von der Datenbank generieren!
  `GENERATED BY DEFAULT AS IDENTITY` sagt dem DBMS, dass wir diese Werte nicht selber angeben wollen...

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
```

Die Spalte `id`

- Wir lassen sie automatisch von der Datenbank generieren!
  `GENERATED BY DEFAULT AS IDENTITY` sagt dem DBMS, dass wir diese Werte nicht selber angeben wollen. . . . . . stattdessen soll es selbstständig einzigartige Werte eintragen.



Die Spalte `id`

- Wir lassen sie automatisch von der Datenbank generieren!
  `GENERATED BY DEFAULT AS IDENTITY` sagt dem DBMS, dass wir diese Werte nicht selber angeben wollen... ...stattdessen soll es selbstständig einzigartige Werte eintragen. Wir brauchen uns also nicht weiter darum zu kümmern.

```
tweise@weise-laptop: ~
tweise@weise-laptop:~$ psql "postgres://boss:superboss123@localhost/factory"
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
```

Interaktive PSQL Session – Teil 2

- Damit ist das SQL-Kommando erklärt.

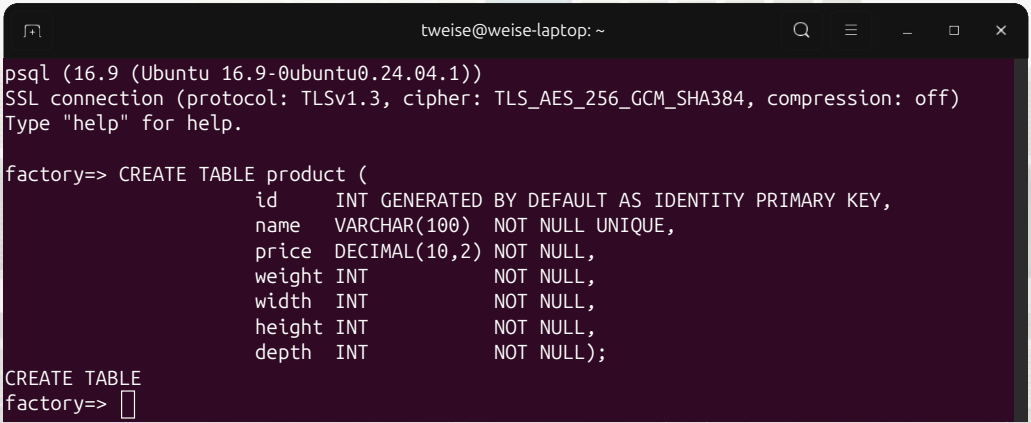
tweise@weise-laptop: ~

```
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
CREATE TABLE
factory=>
```

Interaktive PSQL Session – Teil 2

- Wie bereits gesagt können wir es erfolgreich auf unserem PostgreSQL-Server ausführen.



Interaktive PSQL Session – Teil 2

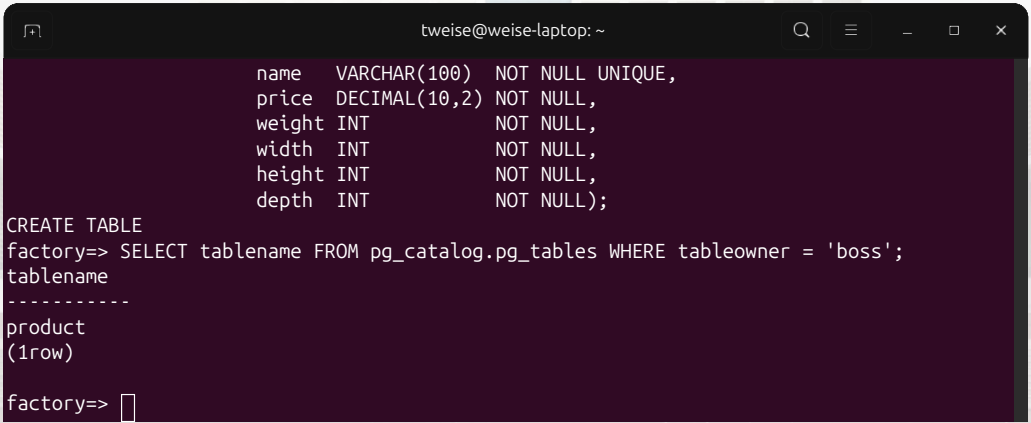
- Mit einer `SELECT ... FROM` Anfrage auf die passende Systemtabelle können wir prüfen, ob die Tabelle `product` nun wirklich existiert.

```
tweise@weise-laptop: ~
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

factory=> CREATE TABLE product (
                      id     INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
                      name   VARCHAR(100)  NOT NULL UNIQUE,
                      price  DECIMAL(10,2) NOT NULL,
                      weight INT           NOT NULL,
                      width  INT           NOT NULL,
                      height INT           NOT NULL,
                      depth  INT           NOT NULL);
CREATE TABLE
factory=> SELECT tablename FROM pg_catalog.pg_tables WHERE tableowner = 'boss';
```

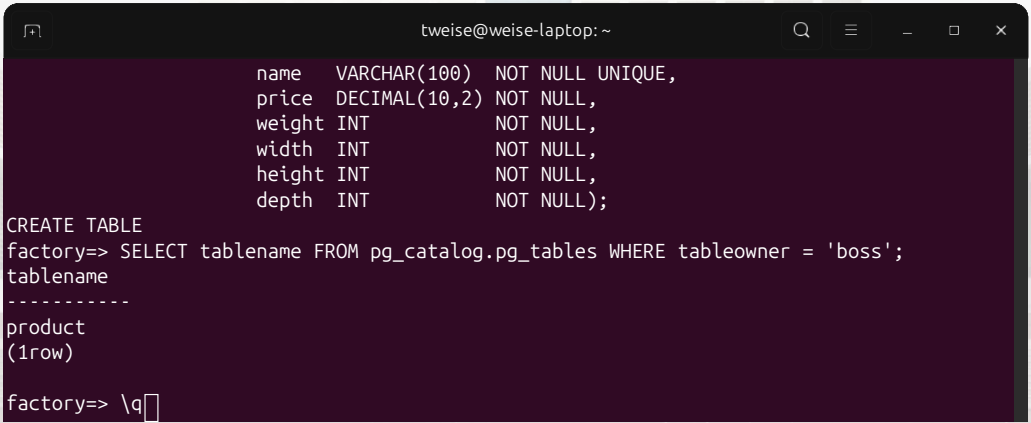
Interaktive PSQL Session – Teil 2

- Mit einer `SELECT ... FROM` Anfrage auf die passende Systemtabelle können wir prüfen, ob die Tabelle `product` nun wirklich existiert. Sie tut es.



Interaktive PSQL Session – Teil 2

- Wir beenden die psql-Session durch `\q` und Enter.



Interaktive PSQL Session – Teil 2

- Wir sind zurück im normalen Terminal.

Erstellen der Tabelle via Script

```sql
/* We create the new table 'product' in our factory database. */

-- List all tables of the user 'boss' in database 'factory'
-- There are no tables yet.
SELECT tablename FROM pg_catalog.pg_tables
    WHERE tableowner='boss';

-- The table 'product' stores all the produces that we can produce.
-- Each row of this table identifies one such product.
CREATE TABLE product (
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,  -- must exist, must be unique
    price DECIMAL(10, 2) NOT NULL,  -- price (RMB): 10 digits, 2 after .
    weight INT NOT NULL,  -- the weight of the product, in grams
    width INT NOT NULL,  -- the width of the product, in mm
    height INT NOT NULL,  -- the height of the product, in mm
    depth INT NOT NULL  -- the depth of the product, in mm
);

-- List all tables of the user 'boss' in database 'factory'
-- Now we see the table 'product'.
SELECT tablename FROM pg_catalog.pg_tables
    WHERE tableowner='boss';
```

```
$ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
    ↪ =1 -ebf create_table_product.sql
 tablename
-----------
(0 rows)

CREATE TABLE
 tablename
-----------
 product
(1 row)

# psql 16.9 succeeded with exit code 0.
```

Erstellen der Tabelle via Script

```sql
/* We create the new table 'product' in our factory database. */

-- List all tables of the user 'boss' in database 'factory'
-- There are no tables yet.
SELECT tablename FROM pg_catalog.pg_tables
    WHERE tableowner='boss';

-- The table 'product' stores all the produces that we can produce.
-- Each row of this table identifies one such product.
CREATE TABLE product (
    id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,  -- must exist, must be unique
    price DECIMAL(10, 2) NOT NULL,  -- price (RMB): 10 digits, 2 after .
    weight INT NOT NULL,  -- the weight of the product, in grams
    width INT NOT NULL,  -- the width of the product, in mm
    height INT NOT NULL,  -- the height of the product, in mm
    depth INT NOT NULL  -- the depth of the product, in mm
);

-- List all tables of the user 'boss' in database 'factory'
-- Now we see the table 'product'.
SELECT tablename FROM pg_catalog.pg_tables
    WHERE tableowner='boss';
```

Erstellen der Tabelle via Script

```
$ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
    ↪ =1 -ebf create_table_product.sql
 tablename
-----------
(0 rows)

CREATE TABLE
 tablename
-----------
 product
(1 row)

# psql 16.9 succeeded with exit code 0.
```

Einfügen von Daten

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.
  - Die kleinsten Schuhe wiegen 1300g und das Gewicht steigt um 25g pro Größe.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.
  - Die kleinsten Schuhe wiegen 1300g und das Gewicht steigt um 25g pro Größe.
- Handtaschen:
  - Handtaschen kommen in den Größen klein (*small*), mittel (*medium*), und groß (*large*).

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.
  - Die kleinsten Schuhe wiegen 1300g und das Gewicht steigt um 25g pro Größe.
- Handtaschen:
  - Handtaschen kommen in den Größen klein (*small*), mittel (*medium*), und groß (*large*).
  - Die entsprechenden Preise sind 100元, 120元, und 150元.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.
  - Die kleinsten Schuhe wiegen 1300g und das Gewicht steigt um 25g pro Größe.
- Handtaschen:
  - Handtaschen kommen in den Größen klein (*small*), mittel (*medium*), und groß (*large*).
  - Die entsprechenden Preise sind 100元, 120元, und 150元.
  - Die kleinste Handtasche passt in einen Schuhkarton, die größeren erfordern größere Boxen.

Daten Einfügen

- Nun haben wir die Tabelle `product` erstellt, aber sie ist leer.
- Füllen wir sie mit Daten!
- Unsere Fabrik hat zwei Produkte: Schuhe (Shoe) und Handtaschen (Purse).
- Schuhe:
  - Schuhe bieten wir in den Schuhgrößen 36 bis 43 an.
  - Ihr Preis beginnt bei 150.99元 für Schuhgröße 36 und steigt um 2元 pro Größe.
  - Die Schuhe passen alle in diese selbe Box.
  - Die kleinsten Schuhe wiegen 1300g und das Gewicht steigt um 25g pro Größe.
- Handtaschen:
  - Handtaschen kommen in den Größen klein (*small*), mittel (*medium*), und groß (*large*).
  - Die entsprechenden Preise sind 100元, 120元, und 150元.
  - Die kleinste Handtasche passt in einen Schuhkarton, die größeren erfordern größere Boxen.
  - Die entsprechenden Gewichte sind dann 500g, 750g, und 1500g.

Zusammenfassung

Zusammenfassung

- Fertig

谢谢您们！

Thank you!

Vielen Dank!

References I

[1] Dirk Angermann. *T-SQL-Abfragen für Microsoft SQL-Server 2022*. Blaufelden, Schwäbisch Hall, Baden-Württemberg, Germany: mitp Verlags GmbH & Co. KG, Juni 2024. ISBN: 978-3-7475-0633-2 (siehe S. 8–13).

[2] Gary Baumgartner, Danny Heap und Richard Krueger. "Numerical Systems". In: *Course Notes for CSC165H: Mathematical Expression and Reasoning for Computer Science*. Toronto, ON, Canada: Department of Computer Science, University of Toronto, Herbst 2006. Kap. 7. URL: https://www.cs.toronto.edu/~krueger/csc165h/f06/lectures/ch7.pdf (besucht am 2024-07-27) (siehe S. 69, 70).

[3] Ben Brumm. "SQL Best Practices and Style Guide". In: *Database Star*. Armadale, VIC, Australia: Elevated Online Services PTY Ltd., 10. Dez. 2024. URL: https://www.databasestar.com/sql-best-practices (besucht am 2025-02-26) (siehe S. 20–26).

[4] Cardinal ("cardinalby"). *Storing Currency Values: Data Types, Caveats, Best Practices*. San Francisco, CA, USA: GitHub Inc, 8. Jan. 2023. URL: https://cardinalby.github.io/blog/post/best-practices/storing-currency-values-data-types (besucht am 2025-02-27) (siehe S. 78–80).

[5] Jonas Gamalielsson und Björn Lundell. "Long-Term Sustainability of Open Source Software Communities beyond a Fork: A Case Study of LibreOffice". In: *8th IFIP WG 2.13 International Conference on Open Source Systems: Long-Term Sustainability OSS'2012*. 10.–13. Sep. 2012, Hammamet, Tunisia. Hrsg. von Imed Hammouda, Björn Lundell, Tommi Mikkonen und Walt Scacchi. Bd. 378. IFIP Advances in Information and Communication Technology (IFIPAICT). Heidelberg, Baden-Württemberg, Germany: Springer-Verlag GmbH Germany, 2012, S. 29–47. ISSN: 1868-4238. ISBN: 978-3-642-33441-2. doi:10.1007/978-3-642-33442-9_3 (siehe S. 8–13).

[6] *LibreOffice – The Document Foundation*. Berlin, Germany: The Document Foundation, 2024. URL: https://www.libreoffice.org (besucht am 2024-12-12) (siehe S. 8–13).

[7] "Numeric Types". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 8.1. URL: https://www.postgresql.org/docs/17/datatype-numeric.html (besucht am 2025-02-27) (siehe S. 79).

[8] Dušan Petković. *Microsoft SQL Server 2019: A Beginner's Guide*. 7. Aufl. New York, NY, USA: McGraw-Hill, Jan. 2020. ISBN: 978-1-260-45888-6 (siehe S. 8–13).

[9] William H. Press, Saul A. Teukolsky, William T. Vetterling und Brian P. Flannery. "1.1 Error, Accuracy, and Stability". In: *Numerical Recipes: The Art of Scientific Computing*. 3. Aufl. Cambridge, England, UK: Cambridge University Press & Assessment, 2007–2011. Kap. 1 Preliminaries, S. 8–12. ISBN: 978-0-521-88068-8. URL: https://numerical.recipes/book.html (besucht am 2024-07-27). Version 3.04 (siehe S. 69, 70).

References II

[10] ."Why not use Double or Float to represent currency?" In: *Stack Overflow*. Hrsg. von Jan Schultke. New York, NY, USA: Stack Exchange Inc., 16. Sep. 2010–13. Jan. 2025. URL: https://stackoverflow.com/questions/3730019 (besucht am 2025-02-27) (siehe S. 71, 72).

[11] Winfried Seimert. *LibreOffice 7.3 – Praxiswissen für Ein- und Umsteiger*. Blaufelden, Schwäbisch Hall, Baden-Württemberg, Germany: mitp Verlags GmbH & Co. KG, Apr. 2022. ISBN: 978-3-7475-0504-5 (siehe S. 8–13).

[12] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence und Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 69, 70).

[13] Randolph West. *How should I store currency values in SQL Server?* Calgary, AB, Canada: Born SQL, 3. Juni 2020. URL: https://bornsql.ca/blog/how-should-i-store-currency-values-in-sql-server (besucht am 2025-02-27) (siehe S. 71, 72, 74, 75, 78–80).

[14] Peter Whyte. *Microsoft SQL Server DBA Blog*. Edinburgh, Scotland, UK, 2018–2025. URL: https://peter-whyte.com/sql-dba-blog (besucht am 2025-06-03) (siehe S. 8–13).