



合肥大學
HEFEI UNIVERSITY



Datenbanken

9. 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

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-Code 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



1. Einleitung
2. Erstellen der Tabelle
3. Einfügen von Daten
4. Daten Selektieren
5. Zusammenfassung





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^{2,58,84} oder LibreOffice Calc^{31,44,67}.

Wir nutzen eine Relationale Datenbank



- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel^{2,58,84} oder LibreOffice Calc^{31,44,67}.
- Auch dort sind Daten in Tabellen organisiert.

Wir nutzen eine Relationale Datenbank



- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel^{2,58,84} oder LibreOffice Calc^{31,44,67}.
- 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^{2,58,84} oder LibreOffice Calc^{31,44,67}.
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können **nur** Text eingeben, der dem vorgegebenen Datentyp entspricht.

Wir nutzen eine Relationale Datenbank



- Wahrscheinlich kennen Sie bereits Tabellenkalkulationssoftware wie Microsoft Excel^{2,58,84} oder LibreOffice Calc^{31,44,67}.
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können **nur** 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^{2,58,84} oder LibreOffice Calc^{31,44,67}.
- Auch dort sind Daten in Tabellen organisiert.
- In einer relationalen Datenbank sind die Spalten der Tabellen aber *typisiert*: Sie können **nur** 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 *Produkte* unseres Betriebs erstellen.

Struktur der Datenbank



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

Ziel



- Das Ziel dieser Einheit ist es, einige grundlegende SQL-Befehle und -Datentypen kennenzulernen.

Ziel



- Das Ziel dieser Einheit ist es, einige grundlegende SQL-Befehle und -Datentypen kennenzulernen.
- Wir lernen, wie man eine Tabelle erstellt.

Ziel



- Das Ziel dieser Einheit ist es, einige grundlegende SQL-Befehle und -Datentypen kennenzulernen.
- Wir lernen, wie man eine Tabelle erstellt.
- Wir lernen eine kleine Untermenge der verfügbaren Datentypen kennen.

Ziel



- Das Ziel dieser Einheit ist es, einige grundlegende SQL-Befehle und -Datentypen kennenzulernen.
- Wir lernen, wie man eine Tabelle erstellt.
- Wir lernen eine kleine Untermenge der verfügbaren Datentypen kennen.
- Wir lernen, wie man Daten in eine Tabelle speichert.

Ziel



- Das Ziel dieser Einheit ist es, einige grundlegende SQL-Befehle und -Datentypen kennenzulernen.
- Wir lernen, wie man eine Tabelle erstellt.
- Wir lernen eine kleine Untermenge der verfügbaren Datentypen kennen.
- Wir lernen, wie man Daten in eine Tabelle speichert.
- Und wir lernen, wie man sie wieder ausliest.



Erstellen der Tabelle



Name der Tabelle



- Das erste, was unsere Tabelle braucht, ist einen Namen.

Name der Tabelle



- Das erste, was unsere Tabelle braucht, ist einen Namen.
- Eine oft verwendete Stilkonvention für Namen für Tabellen besagt:

Name der Tabelle



- Das erste, was unsere Tabelle braucht, ist einen Namen.
- Eine oft verwendete Stilkonvention für Namen für Tabellen besagt:

Gute Praxis

Tabellennamen sollten (Englische) Nomen im Singular sein, mit Kleinbuchstaben geschrieben und ohne Prefix (z. B. kein „tbl_“ am Anfange haben)¹⁵.

Name der Tabelle



- Das erste, was unsere Tabelle braucht, ist einen Namen.
- Eine oft verwendete Stilkonvention für Namen für Tabellen besagt:

Gute Praxis

Tabellennamen sollten (Englische) Nomen im Singular sein, mit Kleinbuchstaben geschrieben und ohne Prefix (z. B. kein „tbl_“ am Anfange haben)¹⁵.

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

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

CREATE TABLE

- Diese Struktur ist im Grunde recht einfach.



CREATE TABLE

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



CREATE TABLE

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



CREATE TABLE



- Nun müssen wir sie in SQL-Kommandos übersetzen.
- Wir müssen dazu Namen, Datentypen und Einschränkungen (EN: *constraints*) für die Spalten festlegen.
- Das SQL-Kommando sieht in etwa so aus:

```
1  -- Erstellen einer neuen Tabelle.
2  --
3  -- tableName: der Name für die zu erstellende Tabelle
4  -- columnX: der Name der X-ten Spalte
5  -- typeOfColumnX: der Datentyp für die X-te Spalte
6  -- constraintsOnColumnX: Einschränkungen für die X-te Spalte, kann weg-
7  -- gelassen werden, oder eine Kombination von
8  -- UNIQUE, NOT NULL, PRIMARY KEY, etc.
9  CREATE TABLE tableName (
10     column1 typeOfColumn1 constraintsOnColumn1,
11     column2 typeOfColumn2 constraintsOnColumn2,
12     column3 typeOfColumn3 constraintsOnColumn3,
13     ...
14 );
```

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 wir den Benutzer `boss` mit Passwort `superboss123` und arbeiten auf der Datenbank `factory`.

```
tweise@weise-laptop: ~  
twaise@weise-laptop: ~$ psql "postgres://boss:superboss123@localhost/factory" █
```

Interaktive PSQL Session – Teil 1



- psql läuft und ist mit dem PostgreSQL DBMS verbunden.

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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 1



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

```
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=> 
```

Kommandostruktur



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

```
tweise@weise-laptop: ~  
twaise@weise-laptop :~$ psql "postgres://boss:superboss123@localhost/factory" █
```

Kommandostruktur



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

```
tweise@weise-laptop: ~  
twaise@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);
```

Kommandostruktur



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

```
tweise@weise-laptop: ~  
twaise@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);
```

Datentypen (points to the data types in the SQL command)

Spalten (points to the column names in the SQL command)

Einschränkungen (points to the constraints in the SQL command)

Die Spalte `name`



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

```
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.

```
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`.

```
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.

```
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



- Jedes Produkt muss einen Namen haben.

```
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



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

```
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



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



- 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.

```
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.

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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.

```
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.

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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 (Datentyp `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 (Datentyp `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^{7,61,81}.

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 (Datentyp `float`) oder Java und C (Datentypen `float`, `double`, ...).
 - Aber Fließkommazahlen sind ungenau!
 - Beispiel: `0.1 + 0.1 + 0.1 - 0.3` ergibt `5.551115123125783e-17` in Python...

Gute Praxis

Nehmen Sie immer an, dass `float`-Werte ungenau sind. Nehmen Sie NIEMALS an, dass diese exact sind^{7,61,81}.

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 (Datentyp `float`) oder Java und C (Datentypen `float`, `double`, ...).
 - Aber Fließkommazahlen sind ungenau!
 - Beispiel: `0.1 + 0.1 + 0.1 - 0.3` ergibt `5.551115123125783e-17` in Python...

Gute Praxis

Repräsentieren Sie niemals Geldwerte mit Fließkommazahlen^{66,82}.

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^{66,82}.

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 fehleranfällig⁸².



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 Datentyp `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 Datentyp `DECIMAL(x, y)` erlaubt es uns, Zahlen mit `x` Stellen (`y` davon nach dem Komma) exakt darzustellen.

Gute Praxis

Geldwerte müssen mit dem Datentyp `DECIMAL` gespeichert werden^{17,82}.

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 Datentyp `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 speichern⁵³.

Gute Praxis

Geldwerte müssen mit dem Datentyp `DECIMAL` gespeichert werden^{17,82}.

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 Datentyp `DECIMAL` gespeichert werden^{17,82}.



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)`.

```
tweise@weise-laptop: ~  
twiese@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

- Jedes Produkt muss einen Preis haben.



```
tweise@weise-laptop: ~  
twaise@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

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



```
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



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

```
tweise@weise-laptop: ~  
twiese@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



- 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.

```
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



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

```
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



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

```
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



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

```
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



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

```
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.

```
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.

```
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.

```
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.

```
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);
```

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* (EN: *primary keys*).

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.

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

```
tweise@weise-laptop: ~  
twaise@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 legen eine zusätzliche Spalte `id` als Primärschlüssel an. Wir markieren die Spalte als `PRIMARY KEY`. Dies impliziert `NOT NULL` und `UNIQUE`.

```
tweise@weise-laptop: ~  
twiese@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 legen eine zusätzliche Spalte `id` als Primärschlüssel an. Wir markieren die Spalte als `PRIMARY KEY`. Dies impliziert `NOT NULL` und `UNIQUE`. Aber was kommt in diese Spalte?

```
tweise@weise-laptop: ~  
twaise@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



- Geben wir ihr ebenfalls den Datentyp `INT`.

```
tweise@weise-laptop: ~  
twaise@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



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

```
tweise@weise-laptop: ~  
twaise@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



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

```
tweise@weise-laptop: ~  
twaise@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



- 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?

```
tweise@weise-laptop: ~  
twaise@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!

```
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...

```
tweise@weise-laptop: ~  
twaise@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.

```
tweise@weise-laptop: ~  
twaise@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. Wir brauchen uns also nicht weiter darum zu kümmern.

```
tweise@weise-laptop: ~  
twaise@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);
```

Ersatzschlüssel (Surrogate Keys)



Gute Praxis

Es ist oftmals eine gute Idee, Ersatzschlüssel (EN: *surrogate (primary) keys*), die auf automatisch erhöhten Ganzzahlen basieren, als Primärschlüssel zu verwenden.¹⁵

Ersatzschlüssel (Surrogate Keys)



Gute Praxis

Es ist oftmals eine gute Idee, Ersatzschlüssel (EN: *surrogate (primary) keys*), die auf automatisch erhöhten Ganzzahlen basieren, als Primärschlüssel zu verwenden.¹⁵

Gute Praxis

Wann immer wir automatisch erhöhte Ganzzahlen als Primärschlüssel für eine Tabelle verwenden, sollten wir sie `id` nennen.

Ersatzschlüssel (Surrogate Keys)



Gute Praxis

Es ist oftmals eine gute Idee, Ersatzschlüssel (EN: *surrogate (primary) keys*), die auf automatisch erhöhten Ganzzahlen basieren, als Primärschlüssel zu verwenden.¹⁵

Gute Praxis

Wann immer wir automatisch erhöhte Ganzzahlen als Primärschlüssel für eine Tabelle verwenden, sollten wir sie `id` nennen.

- Diese Namensgebung ist etwas kontrovers³⁸, hat aber den Vorteil, dass jeder, der unsere Datenbank verwendet, sofort die Bedeutung einer Spalte namens `id` versteht.

Ersatzschlüssel (Surrogate Keys)



Gute Praxis

Es ist oftmals eine gute Idee, Ersatzschlüssel (EN: *surrogate (primary) keys*), die auf automatisch erhöhten Ganzzahlen basieren, als Primärschlüssel zu verwenden.¹⁵

Gute Praxis

Wann immer wir automatisch erhöhte Ganzzahlen als Primärschlüssel für eine Tabelle verwenden, sollten wir sie `id` nennen.

- Diese Namensgebung ist etwas kontrovers³⁸, hat aber den Vorteil, dass jeder, der unsere Datenbank verwendet, sofort die Bedeutung einer Spalte namens `id` versteht.
- Daher ist diese Praxis weit verbreitet^{15,59}.

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.

```
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



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

```
twaise@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.

```
tweise@weise-laptop: ~  
  
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';  
tablename  
-----  
product  
(1row)  
  
factory=> 
```

Interaktive PSQL Session – Teil 2



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

```
tweise@weise-laptop: ~  
  
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';  
tablename  
-----  
product  
(1row)  
  
factory=> \q
```

Interaktive PSQL Session – Teil 2



- Wir sind zurück im normalen Terminal.

```
tweise@weise-laptop: ~  
product  
(1row)  
  
factory=> \q  
tweise@weise-laptop :~$
```

Erstellen der Tabelle via Script



```
1  /* We create the new table 'product' in our factory database. */
2
3  -- List all tables of the user 'boss' in database 'factory'
4  -- There are no tables yet.
5  SELECT tablename FROM pg_catalog.pg_tables
6         WHERE tableowner='boss';
7
8  -- The table 'product' stores all the produces that we can produce.
9  -- Each row of this table identifies one such product.
10 CREATE TABLE product (
11     id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
12     name VARCHAR(100) NOT NULL UNIQUE, -- must exist, must be unique
13     price DECIMAL(10, 2) NOT NULL, -- price (RMB): 10 digits, 2 after .
14     weight INT NOT NULL, -- the weight of the product, in grams
15     width INT NOT NULL, -- the width of the product, in mm
16     height INT NOT NULL, -- the height of the product, in mm
17     depth INT NOT NULL -- the depth of the product, in mm
18 );
19
20 -- List all tables of the user 'boss' in database 'factory'
21 -- Now we see the table 'product'.
22 SELECT tablename FROM pg_catalog.pg_tables
23         WHERE tableowner='boss';
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf create_table_product.sql
2  tablename
3  -----
4  (0 rows)
5
6  CREATE TABLE
7  tablename
8  -----
9  product
10 (1 row)
11
12 # psql 16.11 succeeded with exit code 0.
```

Erstellen der Tabelle via Script



```
1  /* We create the new table 'product' in our factory database. */
2
3  -- List all tables of the user 'boss' in database 'factory'
4  -- There are no tables yet.
5  SELECT tablename FROM pg_catalog.pg_tables
6      WHERE tableowner='boss';
7
8  -- The table 'product' stores all the produces that we can produce.
9  -- Each row of this table identifies one such product.
10 CREATE TABLE product (
11     id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
12     name VARCHAR(100) NOT NULL UNIQUE, -- must exist, must be unique
13     price DECIMAL(10, 2) NOT NULL, -- price (RMB): 10 digits, 2 after .
14     weight INT NOT NULL, -- the weight of the product, in grams
15     width INT NOT NULL, -- the width of the product, in mm
16     height INT NOT NULL, -- the height of the product, in mm
17     depth INT NOT NULL -- the depth of the product, in mm
18 );
19
20 -- List all tables of the user 'boss' in database 'factory'
21 -- Now we see the table 'product'.
22 SELECT tablename FROM pg_catalog.pg_tables
23     WHERE tableowner='boss';
```

Erstellen der Tabelle via Script

```
1 $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
  ↪ =1 -ebf create_table_product.sql
2
3 tablename
4 -----
5 (0 rows)
6
7 CREATE TABLE
8   tablename
9   -----
10  product
11  (1 row)
12
13 # psql 16.11 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.

Daten Einfügen



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

INSERT INTO



- Das SQL-Kommando um Daten an eine Tabelle anzuhängen sieht wie folgt aus:



INSERT INTO



- Das SQL-Kommando um Daten an eine Tabelle anzuhängen sieht wie folgt aus:

```
1  -- Die SQL syntax für das Einfügen/Anhängen von Daten an eine Tabelle.
2  --
3  --      tableName: der Name der Tabelle
4  --      columnI: der Name der I-ten Spalte, wie beim Erstellen
5  --      der Tabelle angegeben
6  --      value_for_rowJcolumnK: der Wert für Spalte K in der J-ten eingefügten
7  --      Spalte
8  --
9  -- Hinweis: Spalten, deren Werte automatisch generiert werden sowie
10 --      Spalten, die *nicht* mit 'NOT NULL' markiert sind müssen
11 --      nicht belegt werden. Sie können sowohl von der Spalten-
12 --      namensliste als auch von den Wertelisten weggelassen werden.
13 INSERT INTO tableName (column1, column2, ...)
14 VALUES (value_for_row1column1, value_for_row1column2, ...),
15         (value_for_row2column1, value_for_row2column2, ...),
16         (value_for_row3column1, value_for_row3column2, ...),
17         ...
18         (value_for_rowNcolumn1, value_for_rowNcolumn2, ...);
```

INSERT INTO



- Das SQL-Kommando um Daten an eine Tabelle anzuhängen sieht wie folgt aus.
- Man kann dieses Kommando beliebig oft verwenden, um Daten anzufügen.

```
1  -- Die SQL syntax für das Einfügen/Anhängen von Daten an eine Tabelle.
2  --
3  -- tableName: der Name der Tabelle
4  -- columnName: der Name der I-ten Spalte, wie beim Erstellen
5  -- der Tabelle angegeben
6  -- value_for_rowJcolumnK: der Wert für Spalte K in der J-ten eingefügten
7  -- Spalte
8  --
9  -- Hinweis: Spalten, deren Werte automatisch generiert werden sowie
10 -- Spalten, die *nicht* mit 'NOT NULL' markiert sind müssen
11 -- nicht belegt werden. Sie können sowohl von der Spalten-
12 -- namensliste als auch von den Wertelisten weggelassen werden.
13 INSERT INTO tableName (column1, column2, ...)
14 VALUES (value_for_row1column1, value_for_row1column2, ...),
15         (value_for_row2column1, value_for_row2column2, ...),
16         (value_for_row3column1, value_for_row3column2, ...),
17         ...
18         (value_for_rowNcolumn1, value_for_rowNcolumn2, ...);
```

Füllen der Tabelle product

```
1  /* Store some data into the table 'product'. */
2
3  -- Print all the contents from table 'product': Nothing.
4  SELECT * FROM product;
5
6  -- Insert 11 products into our table.
7  INSERT INTO product (name, price, weight, width, height, depth)
8  VALUES ('Shoe, Size 36', 150.99, 1300, 350, 250, 130),
9          ('Shoe, Size 37', 152.99, 1325, 350, 250, 130),
10         ('Shoe, Size 38', 154.99, 1350, 350, 250, 130),
11         ('Shoe, Size 39', 156.99, 1375, 350, 250, 130),
12         ('Shoe, Size 40', 158.99, 1400, 350, 250, 130),
13         ('Shoe, Size 41', 160.99, 1425, 350, 250, 130),
14         ('Shoe, Size 42', 162.99, 1450, 350, 250, 130),
15         ('Shoe, Size 43', 164.99, 1475, 350, 250, 130),
16         ('Small Purse', 100, 500, 350, 250, 130),
17         ('Medium Purse', 120, 750, 400, 300, 200),
18         ('Large Purse', 150, 1500, 600, 300, 250);
19
20 -- Now there are 11 rows.
21 SELECT * FROM product;
```

Füllen der Tabelle product

```
1 $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP  
↪ =1 -ebf insert_into_table_product.sql
```

```
2 id | name | price | weight | width | height | depth
```

```
3 -----+-----+-----+-----+-----+-----+-----
```

```
4 (0 rows)
```

```
5  
6 INSERT 0 11
```

```
7 id | name | price | weight | width | height | depth
```

```
8 -----+-----+-----+-----+-----+-----+-----
```

```
9 1 | Shoe, Size 36 | 150.99 | 1300 | 350 | 250 | 130
```

```
10 2 | Shoe, Size 37 | 152.99 | 1325 | 350 | 250 | 130
```

```
11 3 | Shoe, Size 38 | 154.99 | 1350 | 350 | 250 | 130
```

```
12 4 | Shoe, Size 39 | 156.99 | 1375 | 350 | 250 | 130
```

```
13 5 | Shoe, Size 40 | 158.99 | 1400 | 350 | 250 | 130
```

```
14 6 | Shoe, Size 41 | 160.99 | 1425 | 350 | 250 | 130
```

```
15 7 | Shoe, Size 42 | 162.99 | 1450 | 350 | 250 | 130
```

```
16 8 | Shoe, Size 43 | 164.99 | 1475 | 350 | 250 | 130
```

```
17 9 | Small Purse | 100.00 | 500 | 350 | 250 | 130
```

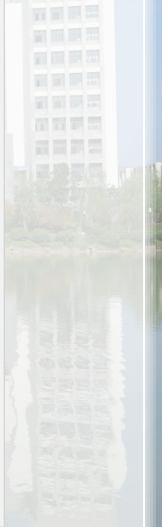
```
18 10 | Medium Purse | 120.00 | 750 | 400 | 300 | 200
```

```
19 11 | Large Purse | 150.00 | 1500 | 600 | 300 | 250
```

```
20 (11 rows)
```

```
21
```

```
22 # psql 16.11 succeeded with exit code 0.
```



UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.
- Wir hatten geschrieben `name VARCHAR(100)NOT NULL UNIQUE`

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.
- Wir hatten geschrieben **name VARCHAR(100)NOT NULL UNIQUE**, was bedeutet das
 - die Spalte **name** Texte speichert, die eine beliebige Länge von bis zu 100 Zeichen haben können (**VARCHAR(100)**),

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.
- Wir hatten geschrieben `name VARCHAR(100)NOT NULL UNIQUE`, was bedeutet das
 - die Spalte `name` Texte speichert, die eine beliebige Länge von bis zu 100 Zeichen haben können (`VARCHAR(100)`),
 - jede Zeile einen Wert in dieser Spalte haben *muss* (`NOT NULL`), und

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.
- Wir hatten geschrieben `name VARCHAR(100)NOT NULL UNIQUE`, was bedeutet das
 - die Spalte `name` Texte speichert, die eine beliebige Länge von bis zu 100 Zeichen haben können (`VARCHAR(100)`),
 - jede Zeile einen Wert in dieser Spalte haben *muss* (`NOT NULL`), und
 - kein Wert in dieser Spalte doppelt auftreten darf (`UNIQUE`).

UNIQUE Einschränkung



- Nun wollen wir noch ausprobieren, ob die **UNIQUE**-Einschränkung wirklich funktioniert.
- Die **UNIQUE**-Einschränkung spezifiziert, dass in der Spalte, auf die es angewendet wird, alle Werte einzigartig sein müssen.
- Wir hatten geschrieben `name VARCHAR(100)NOT NULL UNIQUE`, was bedeutet das
 - die Spalte `name` Texte speichert, die eine beliebige Länge von bis zu 100 Zeichen haben können (`VARCHAR(100)`),
 - jede Zeile einen Wert in dieser Spalte haben *muss* (`NOT NULL`), und
 - kein Wert in dieser Spalte doppelt auftreten darf (`UNIQUE`).
- Was passiert also, wenn wir versuchen, noch einen Eintrag für „Shoe, Size 36“ einzufügen?

Versuch die UNIQUE-Einschränkung zu brechen

```
1  /* Show how the UNIQUE constraint protects our table 'product' */  
2  
3  INSERT INTO product (name, price, weight, width, height, depth)  
4  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP  
   ↳ =1 -ebf insert_into_table_product_error.sql  
2  psql:factory/insert_into_table_product_error.sql:4: ERROR:  duplicate  
   ↳ key value violates unique constraint "product_name_key"  
3  DETAIL:  Key (name)=(Shoe, Size 36) already exists.  
4  psql:factory/insert_into_table_product_error.sql:4: STATEMENT:  /* Show  
   ↳ how the UNIQUE constraint protects our table 'product' */  
5  INSERT INTO product (name, price, weight, width, height, depth)  
6  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);  
7  # psql 16.11 failed with exit code 3.
```

Versuch die UNIQUE-Einschränkung zu brechen

```
1  /* Show how the UNIQUE constraint protects our table 'product' */
2
3  INSERT INTO product (name, price, weight, width, height, depth)
4  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↳ =1 -ebf insert_into_table_product_error.sql
2  psql:factory/insert_into_table_product_error.sql:4: ERROR:  duplicate
   ↳ key value violates unique constraint "product_name_key"
3  DETAIL:  Key (name)=(Shoe, Size 36) already exists.
4  psql:factory/insert_into_table_product_error.sql:4: STATEMENT:  /* Show
   ↳ how the UNIQUE constraint protects our table 'product' */
5  INSERT INTO product (name, price, weight, width, height, depth)
6  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
7  # psql 16.11 failed with exit code 3.
```

Versuch, die UNIQUE-Einschränkung zu brechen

```
1  /* Show how the UNIQUE constraint protects our table 'product' */
2
3  INSERT INTO product (name, price, weight, width, height, depth)
4  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↳ =1 -ebf insert_into_table_product_error.sql
2  psql:factory/insert_into_table_product_error.sql:4: ERROR:  duplicate
   ↳ key value violates unique constraint "product_name_key"
3  DETAIL:  Key (name)=(Shoe, Size 36) already exists.
4  psql:factory/insert_into_table_product_error.sql:4: STATEMENT:  /* Show
   ↳ how the UNIQUE constraint protects our table 'product' */
5  INSERT INTO product (name, price, weight, width, height, depth)
6  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
7  # psql 16.11 failed with exit code 3.
```

- Tatsächlich: Der Versuch, die Einschränkung zu brechen, schlägt fehl.

Versuch, die UNIQUE-Einschränkung zu brechen

```
1  /* Show how the UNIQUE constraint protects our table 'product' */
2
3  INSERT INTO product (name, price, weight, width, height, depth)
4  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↳ =1 -ebf insert_into_table_product_error.sql
2  psql:factory/insert_into_table_product_error.sql:4: ERROR:  duplicate
   ↳ key value violates unique constraint "product_name_key"
3  DETAIL:  Key (name)=(Shoe, Size 36) already exists.
4  psql:factory/insert_into_table_product_error.sql:4: STATEMENT:  /* Show
   ↳ how the UNIQUE constraint protects our table 'product' */
5  INSERT INTO product (name, price, weight, width, height, depth)
6  VALUES ('Shoe, Size 36', 151.99, 1300, 350, 250, 130);
7  # psql 16.11 failed with exit code 3.
```

- Tatsächlich: Der Versuch, die Einschränkung zu brechen, schlägt fehl.
- In einer Microsoft Excel or LibreOffice Calc-Tabelle hätten wir das nicht verhindern können...



Daten Selektieren



Daten Selektieren



- Nun haben wir Daten in die Tabelle `product` eingefügt.

Daten Selektieren



- Nun haben wir Daten in die Tabelle `product` eingefügt.
- Wie bekommen wir sie wieder „heraus“?

Daten Selektieren



- Nun haben wir Daten in die Tabelle `product` eingefügt.
- Wie bekommen wir sie wieder „heraus“?
- Zum Beispiel mit `SELECT * FROM product;`.

Spalten auswählen



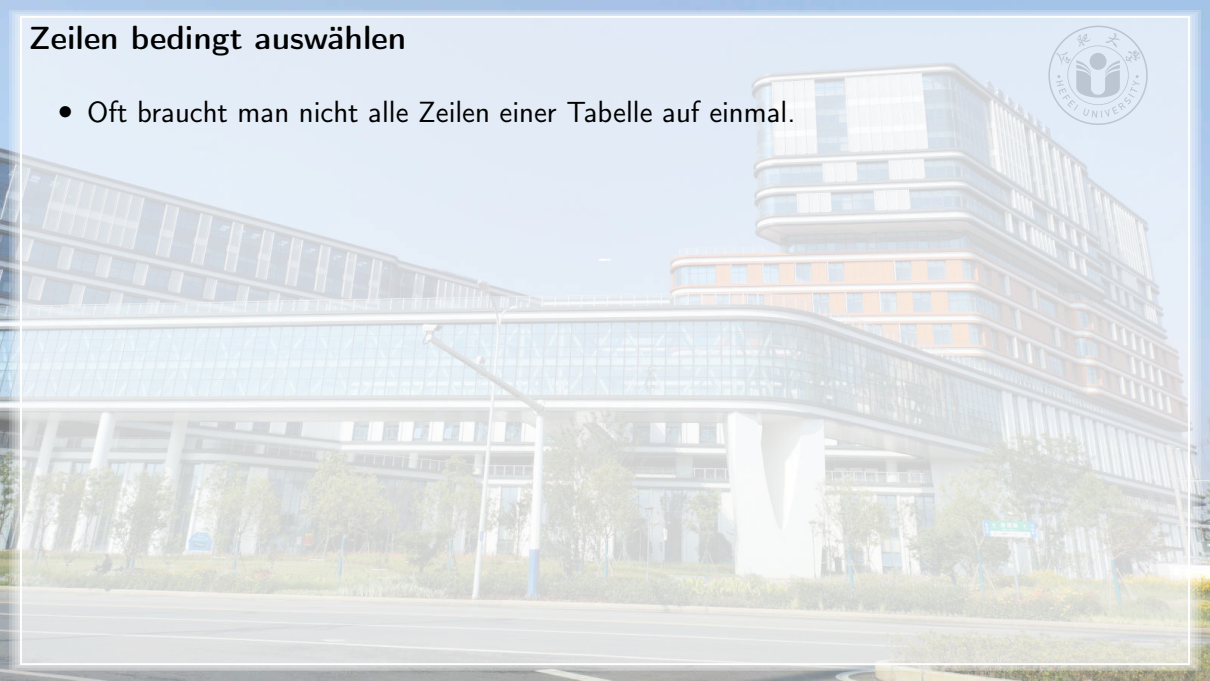
- Hier wir wählen nur ein paar der Spalten aus:

```
1  /* Extract information from the table product. */
2
3  -- List the names and prices of all products.
4  SELECT name, price FROM product;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_1.sql
2      name      | price
3  -----+-----
4  Shoe, Size 36 | 150.99
5  Shoe, Size 37 | 152.99
6  Shoe, Size 38 | 154.99
7  Shoe, Size 39 | 156.99
8  Shoe, Size 40 | 158.99
9  Shoe, Size 41 | 160.99
10 Shoe, Size 42 | 162.99
11 Shoe, Size 43 | 164.99
12 Small Purse   | 100.00
13 Medium Purse  | 120.00
14 Large Purse   | 150.00
15 (11 rows)
16
17 # psql 16.11 succeeded with exit code 0.
```

Zeilen bedingt auswählen

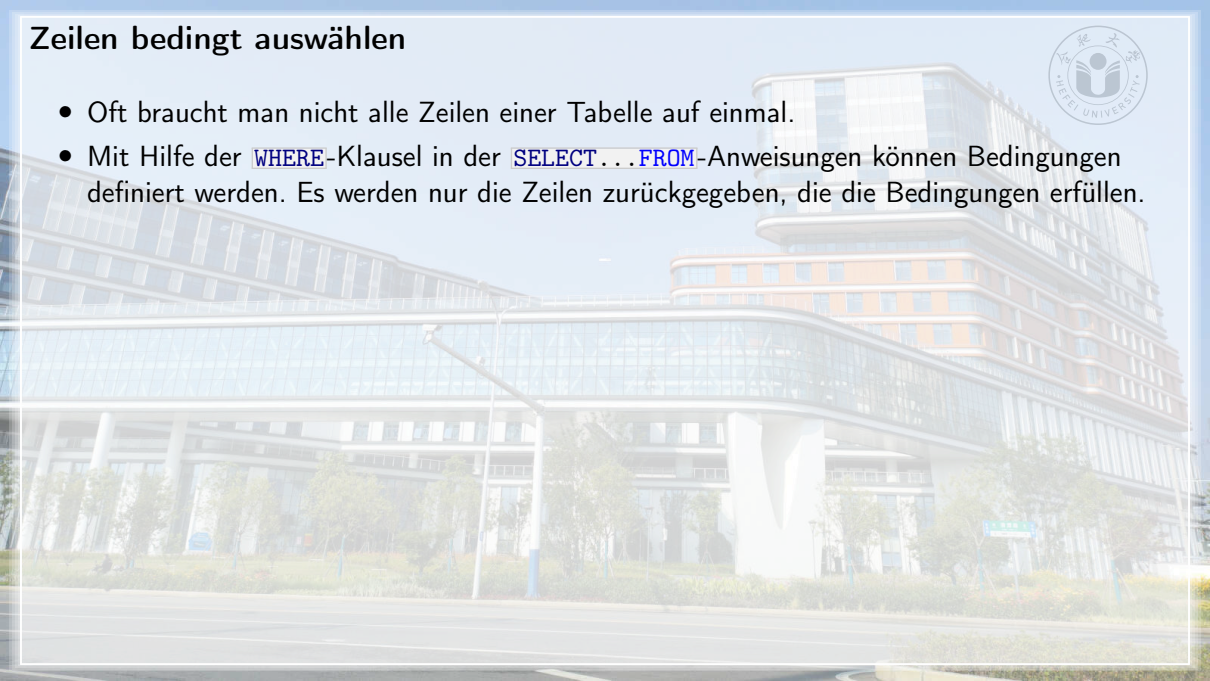
- Oft braucht man nicht alle Zeilen einer Tabelle auf einmal.



Zeilen bedingt auswählen



- Oft braucht man nicht alle Zeilen einer Tabelle auf einmal.
- Mit Hilfe der **WHERE**-Klausel in der **SELECT...FROM**-Anweisungen können Bedingungen definiert werden. Es werden nur die Zeilen zurückgegeben, die die Bedingungen erfüllen.



Zeilen bedingt auswählen



- Oft braucht man nicht alle Zeilen einer Tabelle auf einmal.
- Mit Hilfe der **WHERE**-Klausel in der **SELECT...FROM**-Anweisungen können Bedingungen definiert werden. Es werden nur die Zeilen zurückgegeben, die die Bedingungen erfüllen.

```
1  -- Diese Anweisung wählt nur die Zeilen aus, die die Bedingung erfüllen.  
2  SELECT columnName1, columnName2, ..., columnNameN  
3  FROM tableName  
4  WHERE condition;
```

Zeilen auswählen: Nur Handtaschen



- Nun selektieren wir alle Zeilen, deren `name`-Wert das Wort „Purse“ enthält (via `LIKE`⁵⁶).

```
1  /* Extract information from the table product. */
2
3  -- List the names and prices of all purses.
4  SELECT name, price FROM product WHERE name LIKE '%Purse%';
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_2.sql
2
3      name      | price
4  -----+-----
5  Small Purse   | 100.00
6  Medium Purse  | 120.00
7  Large Purse   | 150.00
8  (3 rows)
9
9  # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen



- Wir können auch in der Abfrage komplett neue Werte mathematisch berechnen⁴⁸ und ihnen mit **AS** einen Spaltennamen zuweise.

```
1  /* Extract information from the table product. */
2
3  -- Compute a value, give it a name, and return it.
4  SELECT name, weight / price AS g_per_yuan FROM product;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_3.sql
2      name      |      g_per_yuan
3  -----+-----
4  Shoe, Size 36 | 8.6098417113716140
5  Shoe, Size 37 | 8.6606967775671613
6  Shoe, Size 38 | 8.7102393702819537
7  Shoe, Size 39 | 8.7585196509331805
8  Shoe, Size 40 | 8.8055852569343984
9  Shoe, Size 41 | 8.8514814584756817
10 Shoe, Size 42 | 8.8962513037609669
11 Shoe, Size 43 | 8.9399357536820413
12 Small Purse   | 5.0000000000000000
13 Medium Purse  | 6.2500000000000000
14 Large Purse   | 10.0000000000000000
15 (11 rows)
16
17 # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen und Sortieren



- Die Abfrageergebnisse mit **ORDER BY** basierend auf den Werten von einer oder mehreren Spalten sortiert werden, wobei **DESC** absteigend und **ASC** für aufsteigendes Sortieren angegeben werden kann.

```
1  /* Extract information from the table product. */
2
3  -- Sort products by "grams per yuan" in a descending fashion.
4  SELECT name, weight / price AS g_per_yuan FROM product
5  ORDER BY g_per_yuan DESC;
```



```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_4.sql
2      name          |      g_per_yuan
3      -----+-----
4  Large Purse       | 10.0000000000000000
5  Shoe, Size 43     | 8.9399357536820413
6  Shoe, Size 42     | 8.8962513037609669
7  Shoe, Size 41     | 8.8514814584756817
8  Shoe, Size 40     | 8.8055852569343984
9  Shoe, Size 39     | 8.7585196509331805
10 Shoe, Size 38     | 8.7102393702819537
11 Shoe, Size 37     | 8.6606967775671613
12 Shoe, Size 36     | 8.6098417113716140
13 Medium Purse      | 6.2500000000000000
14 Small Purse       | 5.0000000000000000
15 (11 rows)
16
17 # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen und Sortieren und Limitieren



- Wir können die Anzahl der zurückgegebenen Zeilen mit **LIMIT** begrenzen.

```
1  /* Extract information from the table product. */
2
3  -- Get the top-five products in terms of grams per yuan.
4  SELECT name, weight / price AS g_per_yuan FROM product
5  ORDER BY g_per_yuan DESC LIMIT 5;
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_5.sql
2
3  name | g_per_yuan
4  -----+-----
5  Large Purse | 10.0000000000000000
6  Shoe, Size 43 | 8.9399357536820413
7  Shoe, Size 42 | 8.8962513037609669
8  Shoe, Size 41 | 8.8514814584756817
9  Shoe, Size 40 | 8.8055852569343984
10 (5 rows)
11 # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen und Sortieren und Limitieren



- Wir können auch Konstanten als Wert einer mit `AS` definierten Spalte zuweise.

```
1  /* Extract information from the table product. */
2
3  -- Get the average price per "shoe" product.
4  SELECT 'Shoe' AS kind, AVG(price) AS mean_price
5         FROM product WHERE name LIKE '%Shoe%';
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_6.sql
2  kind |          mean_price
3  -----+-----
4  Shoe | 157.9900000000000000
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen und Sortieren und Limitieren



- Wir können auch Konstanten als Wert einer mit `AS` definierten Spalte zuweise.
- Statistiken wie der Durchschnitt `AVG(s)` über alle Zeilen können für eine Spalte `s` berechnet werden¹.

```
1  /* Extract information from the table product. */
2
3  -- Get the average price per "shoe" product.
4  SELECT 'Shoe' AS kind, AVG(price) AS mean_price
5         FROM product WHERE name LIKE '%Shoe%';
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_6.sql
2  kind |          mean_price
3  -----+-----
4  Shoe | 157.9900000000000000
5  (1 row)
6
7  # psql 16.11 succeeded with exit code 0.
```

Spalten Erstellen und Sortieren und Limitieren



- Mit `UNION ALL` können die Ausgaben von mehreren `SELECT`-Anweisungen untereinander angereiht werden²².

```
1  /* Extract information from the table product. */
2
3  -- Get the average price for both product types.
4  SELECT 'Shoe' AS kind, AVG(price) AS mean_price
5         FROM product WHERE name LIKE '%Shoe%'
6  UNION ALL SELECT 'Purse' AS kind, AVG(price) AS mean_price
7         FROM product WHERE name LIKE '%Purse%';
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf select_from_table_product_7.sql
2  kind | mean_price
3  -----+-----
4  Shoe  | 157.990000000000000000
5  Purse | 123.3333333333333333
6  (2 rows)
7
8  # psql 16.11 succeeded with exit code 0.
```

Die Schönheit der Datenbanktabellen



- Daten in solchen Tabellen zu verwalten ist auf gewisse Weise wunderschön.



Die Schönheit der Datenbanktabellen



- Daten in solchen Tabellen zu verwalten ist auf gewisse Weise wunderschön.
- Wir können neue Daten in die Tabelle anfügen und doch würden unsere Anfragen funktionieren.

Die Schönheit der Datenbanktabellen



- Daten in solchen Tabellen zu verwalten ist auf gewisse Weise wunderschön.
- Wir können neue Daten in die Tabelle anfügen und doch würden unsere Anfragen funktionieren.
- Ihr Ergebnis wird immer neu berechnet und ist immer aktuell.

Die Schönheit der Datenbanktabellen



- Daten in solchen Tabellen zu verwalten ist auf gewisse Weise wunderschön.
- Wir können neue Daten in die Tabelle anfügen und doch würden unsere Anfragen funktionieren.
- Ihr Ergebnis wird immer neu berechnet und ist immer aktuell.
- Unsere Tabelle könnte hunderte oder zehntausende von Zeilen haben und dennoch würden unsere Anfragen problemlos funktionieren.

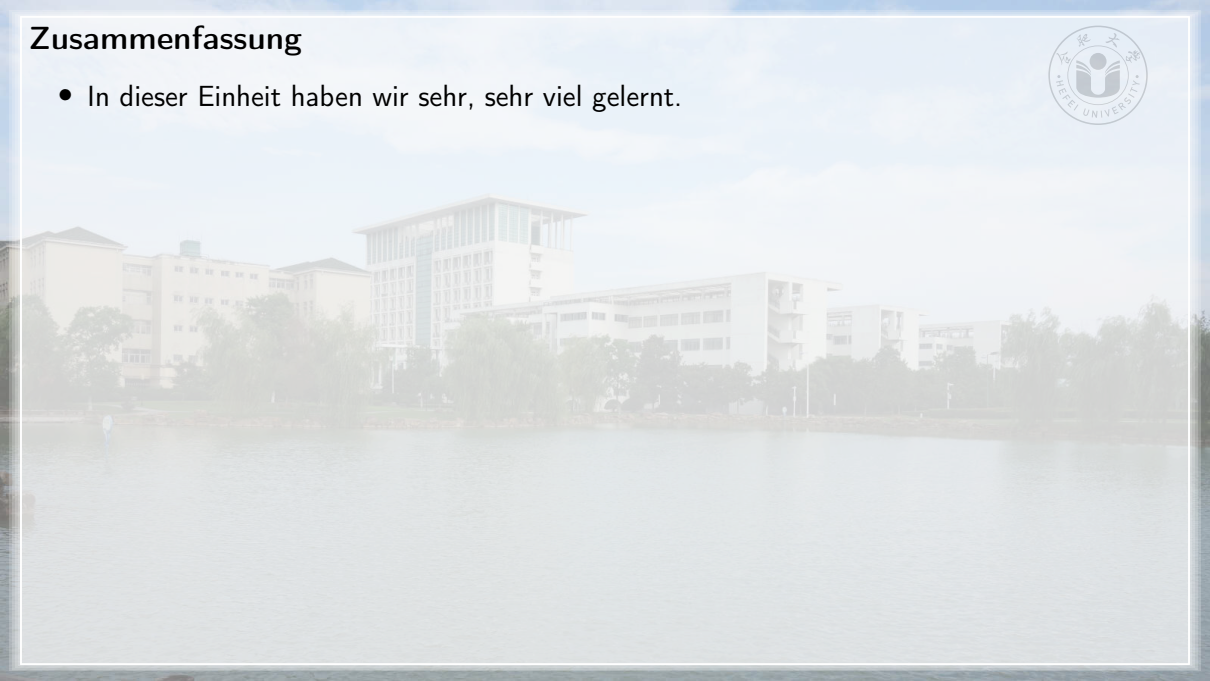


Zusammenfassung



Zusammenfassung

- In dieser Einheit haben wir sehr, sehr viel gelernt.



Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.



Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.

Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.
- Wir haben gelernt, wie man die möglichen Werte für eine Spalte einschränkt, z. B. mit `UNIQUE`, `NOT NULL`, or `PRIMARY KEY`.

Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.
- Wir haben gelernt, wie man die möglichen Werte für eine Spalte einschränkt, z. B. mit `UNIQUE`, `NOT NULL`, or `PRIMARY KEY`.
- Wir haben gelernt, was ein Primärschlüssel (`PRIMARY KEY`) ist, nämlich die Spalte(n), durch deren Wert jede Zeile in einer Tabelle eindeutig identifiziert (und von anderen Tabellen referenziert) werden kann.
- Wir haben gelernt, dass man mit `INSERT INTO` Zeilen an eine Tabelle anhängen kann.

Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.
- Wir haben gelernt, wie man die möglichen Werte für eine Spalte einschränkt, z. B. mit `UNIQUE`, `NOT NULL`, or `PRIMARY KEY`.
- Wir haben gelernt, was ein Primärschlüssel (`PRIMARY KEY`) ist, nämlich die Spalte(n), durch deren Wert jede Zeile in einer Tabelle eindeutig identifiziert (und von anderen Tabellen referenziert) werden kann.
- Wir haben gelernt, dass man mit `INSERT INTO` Zeilen an eine Tabelle anhängen kann.
- Und wir haben gelernt, wie man mit `SELECT FROM` Werte aus einer Tabelle ausliest.

Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.
- Wir haben gelernt, wie man die möglichen Werte für eine Spalte einschränkt, z. B. mit `UNIQUE`, `NOT NULL`, or `PRIMARY KEY`.
- Wir haben gelernt, was ein Primärschlüssel (`PRIMARY KEY`) ist, nämlich die Spalte(n), durch deren Wert jede Zeile in einer Tabelle eindeutig identifiziert (und von anderen Tabellen referenziert) werden kann.
- Wir haben gelernt, dass man mit `INSERT INTO` Zeilen an eine Tabelle anhängen kann.
- Und wir haben gelernt, wie man mit `SELECT FROM` Werte aus einer Tabelle ausliest.
- Wir wissen nun auch, dass man `SELECT FROM` mit weitere Befehle zur Auswahl (`WHERE`, `LIKE`, `LIMIT`), Benennung (`AS`) und Sortierung (`ORDER BY`) sowie zur Berechnung von Statistiken (`AVG`) kombinieren kann.

Zusammenfassung



- In dieser Einheit haben wir sehr, sehr viel gelernt.
- Wir haben gelernt, wie man eine Tabelle in einem SQL-basierten relationalen DBMS erzeugt, nämlich mit `CREATE TABLE`.
- Wir haben mehrere Datentypen kennengelernt, von `INT` über `VARCHAR` bis `DECIMAL`.
- Wir haben gelernt, wie man die möglichen Werte für eine Spalte einschränkt, z. B. mit `UNIQUE`, `NOT NULL`, or `PRIMARY KEY`.
- Wir haben gelernt, was ein Primärschlüssel (`PRIMARY KEY`) ist, nämlich die Spalte(n), durch deren Wert jede Zeile in einer Tabelle eindeutig identifiziert (und von anderen Tabellen referenziert) werden kann.
- Wir haben gelernt, dass man mit `INSERT INTO` Zeilen an eine Tabelle anhängen kann.
- Und wir haben gelernt, wie man mit `SELECT FROM` Werte aus einer Tabelle ausliest.
- Wir wissen nun auch, dass man `SELECT FROM` mit weitere Befehle zur Auswahl (`WHERE`, `LIKE`, `LIMIT`), Benennung (`AS`) und Sortierung (`ORDER BY`) sowie zur Berechnung von Statistiken (`AVG`) kombinieren kann.
- Nicht übel.



谢谢你们！
Thank you!
Vielen Dank!



References I



- [1] "Aggregate Functions". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.21. URL: <https://www.postgresql.org/docs/17/functions-aggregate.html> (besucht am 2025-02-27) (siehe S. 183, 184).
- [2] 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, 211).
- [3] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also⁴ (siehe S. 200, 211).
- [4] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also³ (siehe S. 200, 211).
- [5] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 211, 213).
- [6] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 211).
- [7] 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).
- [8] Ben Beitler. *Hands-On Microsoft Access 2019*. Birmingham, England, UK: Packt Publishing Ltd, März 2020. ISBN: 978-1-83898-747-3 (siehe S. 211).
- [9] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: <https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt> (besucht am 2025-02-05) (siehe S. 213).
- [10] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 210).

References II



- [11] Bernard Obeng Boateng. *Data Modeling with Microsoft Excel*. Birmingham, England, UK: Packt Publishing Ltd, Nov. 2023. ISBN: 978-1-80324-028-2 (siehe S. 211).
- [12] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 212).
- [13] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 212).
- [14] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 210).
- [15] 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. 25–28, 129–132).
- [16] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 211, 212).
- [17] 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).
- [18] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 213).
- [19] Christmas, FL, USA: Simon Sez IT. *Microsoft Access 2021 – Beginner to Advanced*. Birmingham, England, UK: Packt Publishing Ltd, Aug. 2023. ISBN: 978-1-83546-911-8 (siehe S. 211).
- [20] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 213).

References III



- [21] Edgar Frank „Ted“ Codd. “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (besucht am 2025-01-05) (siehe S. 212).
- [22] “Combining Queries (UNION, INTERSECT, EXCEPT)”. In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 7.4. URL: <https://www.postgresql.org/docs/17/queries-union.html> (besucht am 2025-02-27) (siehe S. 185).
- [23] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 213).
- [24] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: <https://dataschool.com/how-to-teach-people-sql> (besucht am 2025-02-27) (siehe S. 213).
- [25] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 213).
- [26] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[?]: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 213).
- [27] Pooyan Doozandeh und Frank E. Ritter. “Some Tips for Academic Writing and Using Microsoft Word”. *XRDS: Crossroads, The ACM Magazine for Students* 26(1):10–11, Herbst 2019. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1528-4972. doi:10.1145/3351470 (siehe S. 212).
- [28] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 211, 212).
- [29] Steve Fanning, Vasudev Narayanan, „flywire“, Olivier Hallot, Jean Hollis Weber, Jenna Sargent, Pulkit Krishna, Dan Lewis, Peter Schofield, Jochen Schiffers, Robert Großkopf, Jost Lange, Martin Fox, Hazel Russman, Steve Schwettman, Alain Romedenne, Andrew Pitonyak, Jean-Pierre Ledure, Drew Jensen und Randolph Gam. *Base Guide 7.3. Revision 1. Based on LibreOffice 7.3 Community*. Berlin, Germany: The Document Foundation, Aug. 2022. URL: <https://books.libreoffice.org/en/BG73/BG73-BaseGuide.pdf> (besucht am 2025-01-13) (siehe S. 211).

References IV



- [30] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: **978-1-83763-564-1** (siehe S. 212).
- [31] 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 (IFIPACT). Berlin/Heidelberg, 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, 211).
- [32] Dawn Griffiths. *Excel Cookbook – Receipts for Mastering Microsoft Excel*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2024. ISBN: **978-1-0981-4332-9** (siehe S. 211).
- [33] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: **978-0-443-23791-1** (siehe S. 212).
- [34] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: **978-0-12-849902-3** (siehe S. 212).
- [35] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: **978-1-0981-0894-6** (siehe S. 211).
- [36] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: **978-0-13-668539-5** (siehe S. 211, 213).
- [37] Manuel Hoffmann, Frank Nagle und Yanuo Zhou. *The Value of Open Source Software*. Working Paper 24-038. Boston, MA, USA: Harvard Business School, 1. Jan. 2024. URL: https://www.hbs.edu/ris/Publication%20Files/24-038_51f8444f-502c-4139-8bf2-56eb4b65c58a.pdf (besucht am 2025-06-04) (siehe S. 212).
- [38] "Why is naming a table's Primary Key column „Id“ considered bad practice? [closed]". In: *Software Engineering*. Hrsg. von Jean-Philippe Leclerc. New York, NY, USA: Stack Exchange Inc., 17. Okt. 2011–22. Dez. 2024. URL: <https://softwareengineering.stackexchange.com/questions/114728> (besucht am 2025-02-27) (siehe S. 129–132).

References V



- [39] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 212).
- [40] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: [https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_\(en\).zip](https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip) (besucht am 2025-01-08). Consists of several parts, see <https://modern-sql.com/standard> for information where to obtain them. (Siehe S. 213).
- [41] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 212).
- [42] Joan Lambert und Curtis Frye. *Microsoft Office Step by Step (Office 2021 and Microsoft 365)*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Juni 2022. ISBN: 978-0-13-754493-6 (siehe S. 211).
- [43] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 212).
- [44] *LibreOffice – The Document Foundation*. Berlin, Germany: The Document Foundation, 2024. URL: <https://www.libreoffice.org> (besucht am 2024-12-12) (siehe S. 8–13, 211).
- [45] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: <https://www.britannica.com/technology/client-server-architecture> (besucht am 2025-01-20) (siehe S. 210).
- [46] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 212).
- [47] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 211).
- [48] "Mathematical Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.3. URL: <https://www.postgresql.org/docs/17/functions-math.html> (besucht am 2025-02-27) (siehe S. 180).

References VI



- [49] Ron McFadyen und Cindy Miller. *Relational Databases and Microsoft Access*. 3. Aufl. Palatine, IL, USA: Harper College, 2014–2019. URL: <https://harpercollege.pressbooks.pub/relationaldatabases> (besucht am 2025-04-11) (siehe S. 211).
- [50] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 213).
- [51] *Microsoft Word*. Redmond, WA, USA: Microsoft Corporation, 2024. URL: <https://www.microsoft.com/en-us/microsoft-365/word> (besucht am 2024-12-12) (siehe S. 212).
- [52] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 210).
- [53] "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).
- [54] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 212).
- [55] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 210).
- [56] "Pattern Matching". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.7. URL: <https://www.postgresql.org/docs/17/functions-matching.html> (besucht am 2025-02-27) (siehe S. 179).
- [57] Yasset Pérez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, Tobias Ternent, Stephen J. Eglen, Daniel S. Katz, Tom J. Pollard, Alexander Kononov, Robert M. Flight, Kai Blin und Juan Antonio Vizcaíno. "Ten Simple Rules for Taking Advantage of Git and GitHub". *PLOS Computational Biology* 12(7), 14. Juli 2016. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1553-7358. doi:10.1371/JOURNAL.PCBI.1004947 (siehe S. 210).
- [58] 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, 211).

References VII



- [59] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: <https://www.postgresql.org/docs/17/index.html> (besucht am 2025-02-25) (siehe S. 129–132).
- [60] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 212).
- [61] 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 (CUP), 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).
- [62] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 210).
- [63] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 211).
- [64] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021)*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 212).
- [65] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 210).
- [66] "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).
- [67] 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, 211).

References VIII



- [68] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 211).
- [69] Anna Skoulikari. *Learning Git*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mai 2023. ISBN: 978-1-0981-3391-7 (siehe S. 210).
- [70] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 212).
- [71] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: <https://www.postgresql.org/docs/17/sql-commands.html> (besucht am 2025-02-25) (siehe S. 213).
- [72] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 207, 213).
- [73] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burghthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of⁷² (siehe S. 213).
- [74] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 212, 213).
- [75] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 212).
- [76] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 211).
- [77] Mariot Tsitoara. *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*. New York, NY, USA: Apress Media, LLC, März 2024. ISBN: 979-8-8688-0215-7 (siehe S. 210, 213).

References IX



- [78] Laurie A. Ulrich und Ken Cook. *Access For Dummies*. Hoboken, NJ, USA: For Dummies (Wiley), Dez. 2021. ISBN: 978-1-119-82908-9 (siehe S. 211).
- [79] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 211).
- [80] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: <https://thomasweise.github.io/databases> (besucht am 2025-01-05) (siehe S. 210–212).
- [81] 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 and 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, 212).
- [82] 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).
- [83] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: <https://www.ibm.com/think/topics/relational-databases> (besucht am 2025-01-05) (siehe S. 212).
- [84] 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, 211).
- [85] Ulf Michael „Monty“ Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 212).
- [86] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: <https://www.techtarget.com/searchdatamanagement/definition/database-management-system> (besucht am 2025-01-11) (siehe S. 210).

References X



- [87] Pavlo V. Zahorodko und Pavlo V. Merzlykin. "An Approach for Processing and Document Flow Automation for Microsoft Word and LibreOffice Writer File Formats". In: *4th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW'2021)*. 18. Dez. 2021, Virtual Event and Kryvyi Rih, Ukraine. Hrsg. von Arnold E. Kiv, Serhiy O. Semerikov, Vladimir N. Soloviev und Andrii M. Striuk. Bd. 3077 der Reihe CEUR Workshop Proceedings ([CEUR-WS.org](https://ceur-ws.org)). Aachen, Nordrhein-Westfalen, Germany: CEUR-WS Team, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, 2022, S. 66–82. ISSN: **1613-0073**. URL: <https://ceur-ws.org/Vol-3077/paper12.pdf> (besucht am 2025-10-04) (siehe S. **211**, **212**).
- [88] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: **978-1-78439-687-9** (siehe S. **210**).

Glossary (in English) I



Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs^{14,52,88}. Learn more at <https://www.gnu.org/software/bash>.

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients^{10,45,55,62,65}. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of database (DB) software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*⁸⁰.

DBA A *database administrator* is the person or group responsible for the effective use of database technology in an organization or enterprise.

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB⁸⁶.

Git is a distributed Version Control Systems (VCS) which allows multiple users to work on the same code while preserving the history of the code changes^{69,77}. Learn more at <https://git-scm.com>.

GitHub is a website where software projects can be hosted and managed via the Git VCS^{57,77}. Learn more at <https://github.com>.

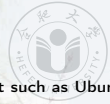
IT information technology

Glossary (in English) II



- LAMP Stack** A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP^{16,36}.
- LibreOffice** is an open source office suite^{31,44,67} which is a good and free alternative to Microsoft Office. It offers software such as LibreOffice Writer, LibreOffice Calc, and LibreOffice Base. See⁸⁰ for more information and installation instructions.
- LibreOffice Base** is a DBMS that can work on stand-alone files but also connect to other popular relational databases^{29,67}. It is part of LibreOffice^{31,44,67} and has functionality that is comparable to Microsoft Access^{8,19,78}.
- LibreOffice Calc** is a spreadsheet software that allows you to arrange and perform calculations with data in a tabular grid. It is a free and open source spreadsheet software^{44,67}, i.e., an alternative to Microsoft Excel. It is part of LibreOffice^{31,44,67}.
- LibreOffice Writer** is a free and open source text writing program⁸⁷ and part of LibreOffice^{31,44,67}. It is a good alternative to Microsoft Word.
- Linux** is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{5,35,68,76,79}. We recommend using it for this course, for software development, and for research. Learn more at <https://www.linux.org>. Its variant Ubuntu is particularly easy to use and install.
- MariaDB** An open source relational database management system that has forked off from MySQL^{3,4,6,28,47,63}. See <https://mariadb.org> for more information.
- Microsoft Access** is a DBMS that can work on DBs stored in single, stand-alone files but also connect to other popular relational databases^{8,19,49,78}. It is part of Microsoft Office. A free and open source alternative to this commercial software is LibreOffice Base.
- Microsoft Excel** is a spreadsheet program that allows users to store, organize, manipulate, and calculate data in tabular structures^{11,32,42}. It is part of Microsoft Office. A free alternative to this commercial software is LibreOffice Calc^{44,67}.
- Microsoft Office** is a commercial suite of office software, including Microsoft Excel, Microsoft Word, and Microsoft Access⁴². LibreOffice is a free and open source alternative.
- Microsoft SQL Server** The Microsoft SQL Server is a successful commercial relational/Structured Query Language (SQL)-based DBMS^{2,58,84}. Learn more at <https://www.microsoft.com/sql-server> and <https://learn.microsoft.com/en-us/sql>.

Glossary (in English) III



- Microsoft Windows is a commercial proprietary operating system¹³. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at <https://www.microsoft.com/windows>.
- Microsoft Word is one of the leading text writing programs^{27,51,87} and part of Microsoft Office. A free alternative to this commercial software is the LibreOffice Writer.
- MySQL An open source relational database management system^{12,28,64,75,85}. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.
- OSS Open source software, i.e., software that can freely be used, whose source code is made available in the internet, and which is usually developed cooperatively over the internet as well³⁷. Typical examples are Python, Linux, Git, and PostgreSQL.
- PostgreSQL An open source object-relational DBMS^{30,54,60,75}. See <https://postgresql.org> for more information.
- psql is the client program used to access the PostgreSQL DBMS server.
- Python The Python programming language^{39,43,46,81}, i.e., what you will learn about in our book⁸¹. Learn more at <https://python.org>.
- relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other^{21,33,34,70,74,80,83}.
- server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carrying the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers¹⁶ in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“⁴¹.

Glossary (in English) IV



SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{18,23–25,40,50,71–74}. It is understood by many DBMSes. You find the SQL commands supported by PostgreSQL in the reference⁷¹.

terminal A terminal is a text-based window where you can enter commands and execute them^{5,20}. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf  + , dann Schreiben von `cmd`, dann Druck auf . Under Ubuntu Linux,  +  +  opens a terminal, which then runs a Bash shell inside.

Ubuntu is a variant of the open source operating system Linux^{20,36}. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at <https://ubuntu.com>. If you are in China, you can download it from <https://mirrors.ustc.edu.cn/ubuntu-releases>.

VCS A *Version Control System* is a software which allows you to manage and preserve the historical development of your program code⁷⁷. A distributed VCS allows multiple users to work on the same code and upload their changes to the server, which then preserves the change history. The most popular distributed VCS is Git.

WWW World Wide Web^{9,26}