



合肥大學
HEFEI UNIVERSITY



Datenbanken

11. Fabrik-Datenbank: Tabelle *demand*

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-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielen finden Sie unter <https://github.com/thomasWeise/databasesCode>.



Outline



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





Einleitung





- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.

Einleitung



- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.
- Als nächstes wollen wir auch Bestellungen speichern können.

Einleitung



- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.
- Als nächstes wollen wir auch Bestellungen speichern können.
- Wir wählen dafür ein extrem simplifiziertes Konzept.

Einleitung



- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.
- Als nächstes wollen wir auch Bestellungen speichern können.
- Wir wählen dafür ein extrem simplifiziertes Konzept Ein Kunde kann mit einer Bestellung eine Menge von genau einem Produkt ordern.



- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.
- Als nächstes wollen wir auch Bestellungen speichern können.
- Wir wählen dafür ein extrem simplifiziertes Konzept Ein Kunde kann mit einer Bestellung eine Menge von genau einem Produkt ordern.
- In echten Online-Shops können Bestellungen aus mehreren Produkten bestehen, Zahlungsoptionen haben und vielleicht auch Lieferadressen. . .

- Jetzt können wir sowohl Kunden als auch Produkte in unserer Datenbank speichern.
- Als nächstes wollen wir auch Bestellungen speichern können.
- Wir wählen dafür ein extrem simplifiziertes Konzept Ein Kunde kann mit einer Bestellung eine Menge von genau einem Produkt ordern.
- In echten Online-Shops können Bestellungen aus mehreren Produkten bestehen, Zahlungsoptionen haben und vielleicht auch Lieferadressen. . . . wir wollen unser Beispiel aber schön einfach halten.



Erstellen der Tabelle



Grobstruktur

- Wir nennen unsere Tabelle für Bestellungen `demand`.



Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .

Gute Praxis

Never use Structured Query Language (SQL) keywords or reserved words as names, e.g., for columns or tables¹⁰.

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .
- Die Tabelle muss für jede Bestellung folgende Daten speichern

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .
- Die Tabelle muss für jede Bestellung folgende Daten speichern:
 - den Kunde, der die Bestellung aufgibt

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .
- Die Tabelle muss für jede Bestellung folgende Daten speichern:
 - den Kunde, der die Bestellung aufgibt,
 - das bestellte Produkt

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .
- Die Tabelle muss für jede Bestellung folgende Daten speichern:
 - den Kunde, der die Bestellung aufgibt,
 - das bestellte Produkt,
 - die Anzahl der bestellten Einheiten des Produktes

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`. Ich würde sie lieber `order` nennen, aber das ist ja ein SQL-Schlüsselwort. . .
- Die Tabelle muss für jede Bestellung folgende Daten speichern:
 - den Kunde, der die Bestellung aufgibt,
 - das bestellte Produkt,
 - die Anzahl der bestellten Einheiten des Produktes, und
 - und das Datum, an dem die Bestellung aufgegeben wurde.

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`.
- Insgesamt könnten die Daten also wie folgt aussehen:

Grobstruktur



- Wir nennen unsere Tabelle für Bestellungen `demand`.
- In den Spalten für Kunde und Produkt werden die Primärschlüssel aus den entsprechenden Tabellen stehen. . .

id	customer	product	amount	ordered
1	1 (Bibbo)	7 (Shoe, Size 42)	12	2024-11-21
2	2 (Bebbo)	3 (Shoe, Size 38)	2	2024-12-09
3	3 (Bebba)	2 (Shoe, Size 37)	7	2024-12-16
4	2 (Bebbo)	5 (Shoe, Size 40)	7	2024-12-30
5	1 (Bibbo)	5 (Shoe, Size 40)	3	2025-01-05
6	2 (Bebbo)	6 (Shoe, Size 41)	4	2025-01-12
7	3 (Bebba)	11 (Large Purse)	10	2025-01-16
8	2 (Bebbo)	3 (Shoe, Size 38)	6	2025-02-05
...	...	...	...	...

Erstellen der Tabelle mit CREATE TABLE

- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.



Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT    NOT NULL REFERENCES customer(id),
7      product     INT    NOT NULL REFERENCES product(id),
8      amount      INT    NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.

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

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.
- Wir haben dieses Kommando via psql an den PostgreSQL-Server geschickt und das Erstellen der Tabelle hat funktioniert.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT NOT NULL REFERENCES customer(id),
7      product     INT NOT NULL REFERENCES product(id),
8      amount      INT NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.
- Wir haben dieses Kommando via psql an den PostgreSQL-Server geschickt und das Erstellen der Tabelle hat funktioniert.
- Wieder gibt es einige interessante neue Komponenten.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT NOT NULL REFERENCES customer(id),
7      product     INT NOT NULL REFERENCES product(id),
8      amount      INT NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.
- Wir haben dieses Kommando via psql an den PostgreSQL-Server geschickt und das Erstellen der Tabelle hat funktioniert.
- Wieder gibt es einige interessante neue Komponenten: Was bedeutet `REFERENCES`?

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT  NOT NULL REFERENCES customer(id),
7      product     INT  NOT NULL REFERENCES product(id),
8      amount      INT  NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.
- Wir haben dieses Kommando via psql an den PostgreSQL-Server geschickt und das Erstellen der Tabelle hat funktioniert.
- Wieder gibt es einige interessante neue Komponenten: Was bedeutet `REFERENCES`? Was ist ein `DATE`?

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id            INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer INT NOT NULL REFERENCES customer(id),
7      product  INT NOT NULL REFERENCES product(id),
8      amount   INT NOT NULL,
9      ordered  DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Erstellen der Tabelle mit CREATE TABLE



- Wir wissen ja schon, dass man Tabellen mit `CREATE TABLE` erstellt.
- Wir haben dieses Kommando via psql an den PostgreSQL-Server geschickt und das Erstellen der Tabelle hat funktioniert.
- Wieder gibt es einige interessante neue Komponenten: Was bedeutet `REFERENCES`? Was ist ein `DATE`? Schauen wir uns das Schritt für Schritt an.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id            INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer INT NOT NULL REFERENCES customer(id),
7      product  INT NOT NULL REFERENCES product(id),
8      amount   INT NOT NULL,
9      ordered  DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.



Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.
- In anderen Worten

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.
- In anderen Worten:
 - ihre Werte sind ganzzahlig (`INT`)

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.
- In anderen Worten:
 - ihre Werte sind ganzzahlig (`INT`),
 - werden automatisch von einer automatisch inkrementierten Sequenz generiert (`GENERATED BY DEFAULT AS IDENTITY`)

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.
- In anderen Worten:
 - ihre Werte sind ganzzahlig (`INT`),
 - werden automatisch von einer automatisch inkrementierten Sequenz generiert (`GENERATED BY DEFAULT AS IDENTITY`), und
 - stellen den Primärschlüssel für unsere Tabelle dar.

Die Spalte id



- Die erste Spalte nennen wir wieder `id`.
- Sie hat den Typ `INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY`.
- In anderen Worten:
 - ihre Werte sind ganzzahlig (`INT`),
 - werden automatisch von einer automatisch inkrementierten Sequenz generiert (`GENERATED BY DEFAULT AS IDENTITY`), und
 - stellen den Primärschlüssel für unsere Tabelle dar.
- Genaugenommen ist die Spalte wieder ein sogenannter *Surrogate Key*, zu Deutsch ein Ersatzschlüssel.

Die Spalte customer

- Zu jeder Bestellung gehört ein Kunde.



Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.
- Die Daten der Kunden sind in der Tabelle `customer` gespeichert.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.
- Die Daten der Kunden sind in der Tabelle `customer` gespeichert.
- Sie werden über den Primärschlüssel identifiziert, den wir `id` genannt haben.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.
- Die Daten der Kunden sind in der Tabelle `customer` gespeichert.
- Sie werden über den Primärschlüssel identifiziert, den wir `id` genannt haben.
- Die Werte für `id` sind einzigartig in der Tabelle `customer`: Jeder Record = jede Zeile der Tabelle hat einen anderen `id`-Wert.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.
- Die Daten der Kunden sind in der Tabelle `customer` gespeichert.
- Sie werden über den Primärschlüssel identifiziert, den wir `id` genannt haben.
- Die Werte für `id` sind einzigartig in der Tabelle `customer`: Jeder Record = jede Zeile der Tabelle hat einen anderen `id`-Wert.
- Wir könnten also für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- In der zweiten Spalte, die wir `customer` nennen, wollen wir also die Kunden-Einträge referenzieren.
- Die Daten der Kunden sind in der Tabelle `customer` gespeichert.
- Sie werden über den Primärschlüssel identifiziert, den wir `id` genannt haben.
- Die Werte für `id` sind einzigartig in der Tabelle `customer`: Jeder Record = jede Zeile der Tabelle hat einen anderen `id`-Wert.
- Wir könnten also für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).
- Wie stellen wir sicher, dass die Zahlen, die wir in der Spalte `customer` unserer neuen Tabelle `demand` speichern, wirklich passende Kunden-`ids` sind?

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).
- Wie stellen wir sicher, dass die Zahlen, die wir in der Spalte `customer` unserer neuen Tabelle `demand` speichern, wirklich passende Kunden-`ids` sind?
- Irgendwie müssen wir dem DBMS sagen: „Diese neue Spalte hier referenziert die Spalte `id` der Tabelle `customer`. Es dürfen nur Werte auftreten, die auch dort vorkommen!“

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle speichern.
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).
- Wie stellen wir sicher, dass die Zahlen, die wir in der Spalte `customer` unserer neuen Tabelle `demand` speichern, wirklich passende Kunden-`ids` sind?
- Irgendwie müssen wir dem DBMS sagen: „Diese neue Spalte hier referenziert die Spalte `id` der Tabelle `customer`. Es dürfen nur Werte auftreten, die auch dort vorkommen!“
- Das geht, in dem wir `REFERENCES customer(id)` schreiben, wobei sich `customer` auf die Tabelle `customer` und `id` auf die Spalte `id` der Tabelle `customer` bezieht.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).
- Wie stellen wir sicher, dass die Zahlen, die wir in der Spalte `customer` unserer neuen Tabelle `demand` speichern, wirklich passende Kunden-`ids` sind?
- Irgendwie müssen wir dem DBMS sagen: „Diese neue Spalte hier referenziert die Spalte `id` der Tabelle `customer`. Es dürfen nur Werte auftreten, die auch dort vorkommen!“
- Das geht, in dem wir `REFERENCES customer(id)` schreiben, wobei sich `customer` auf die Tabelle `customer` und `id` auf die Spalte `id` der Tabelle `customer` bezieht.
- Durch `REFERENCES customer(id)` wird die neue Spalte zum sogenannten Fremdschlüssel (Foreign Key).

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Kunden-Record in der Tabelle `customer` finden und z. B. den Namen und die Adresse auslesen.
- Wir würden also eine neue Spalte namens `customer` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `customer` ist ja auch ein `INT`).
- Wie stellen wir sicher, dass die Zahlen, die wir in der Spalte `customer` unserer neuen Tabelle `demand` speichern, wirklich passende Kunden-`ids` sind?
- Irgendwie müssen wir dem DBMS sagen: „Diese neue Spalte hier referenziert die Spalte `id` der Tabelle `customer`. Es dürfen nur Werte auftreten, die auch dort vorkommen!“
- Das geht, in dem wir `REFERENCES customer(id)` schreiben, wobei sich `customer` auf die Tabelle `customer` und `id` auf die Spalte `id` der Tabelle `customer` bezieht.
- Durch `REFERENCES customer(id)` wird die neue Spalte zum sogenannten Fremdschlüssel (Foreign Key).
- Zusätzlich muss die Spalte als `NOT NULL` markiert werden.

Die Spalte customer



- Zu jeder Bestellung gehört ein Kunde.
- Wir könnten für jede Bestellung den `id`-Wert des Kunden-Eintrags aus der `customer`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT NOT NULL REFERENCES customer(id),
7      product     INT NOT NULL REFERENCES product(id),
8      amount      INT NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Die Spalte product

- Zu jeder Bestellung gehört ein Produkt.



Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Produkt-Record in der Tabelle `product` finden und z. B. den Preis auslesen.

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Produkt-Record in der Tabelle `product` finden und z. B. den Preis auslesen.
- Wir würden also eine neue Spalte namens `product` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `product` ist ja auch ein `INT`).

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Produkt-Record in der Tabelle `product` finden und z. B. den Preis auslesen.
- Wir würden also eine neue Spalte namens `product` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `product` ist ja auch ein `INT`).
- Dazu müssen wir `REFERENCES product(id)` schreiben, wobei sich `product` auf die Tabelle `product` und `id` auf die Spalte `id` der Tabelle `product` bezieht.

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Produkt-Record in der Tabelle `product` finden und z. B. den Preis auslesen.
- Wir würden also eine neue Spalte namens `product` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `product` ist ja auch ein `INT`).
- Dazu müssen wir `REFERENCES product(id)` schreiben, wobei sich `product` auf die Tabelle `product` und `id` auf die Spalte `id` der Tabelle `product` bezieht.
- Dadurch wird sichergestellt, dass es nur Bestellungen geben kann, deren `product`-Wert zu einer Zeile in der Tabelle `product` passt.

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².
- Mit diesem Wert können wir dann den Produkt-Record in der Tabelle `product` finden und z. B. den Preis auslesen.
- Wir würden also eine neue Spalte namens `product` in der Tabelle `demand` erstellen, die vom Datentyp `INT` ist (denn die Spalte `id` in der Tabelle `product` ist ja auch ein `INT`).
- Dazu müssen wir `REFERENCES product(id)` schreiben, wobei sich `product` auf die Tabelle `product` und `id` auf die Spalte `id` der Tabelle `product` bezieht.
- Dadurch wird sichergestellt, dass es nur Bestellungen geben kann, deren `product`-Wert zu einer Zeile in der Tabelle `product` passt.
- Zusätzlich muss die Spalte als `NOT NULL` markiert werden.

Die Spalte product



- Zu jeder Bestellung gehört ein Produkt.
- Wir könnten für jede Bestellung den `id`-Wert des Produkt-Eintrags aus der `product`-Tabelle als Fremdschlüssel (Foreign Key) speichern²².

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT    NOT NULL REFERENCES customer(id),
7      product     INT    NOT NULL REFERENCES product(id),
8      amount      INT    NOT NULL,
9      ordered     DATE  NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Die Spalte amount

- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.



Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.

Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.

Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.
- Stellt das aber sicher, dass die Menge „richtig“ ist?

Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.
- Stellt das aber sicher, dass die Menge „richtig“ ist?
- Nein. Wir könnten 0 als Menge eingeben. Oder -5.

Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.
- Stellt das aber sicher, dass die Menge „richtig“ ist?
- Nein. Wir könnten 0 als Menge eingeben. Oder -5.
- Wir fügen also ein `CONSTRAINT` hinzu, das verlangt `(amount > 0) AND (amount < 1000000)`.

Die Spalte amount



- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.
- Stellt das aber sicher, dass die Menge „richtig“ ist?
- Nein. Wir könnten 0 als Menge eingeben. Oder -5.
- Wir fügen also ein `CONSTRAINT` hinzu, das verlangt `(amount > 0) AND (amount < 1000000)`.
- Randnotiz: Solche Einschränkungen wären auch für die Tabelle `product` sinnvoll gewesen, allerdings war diese Leereinheit schon kompliziert genug...



Die Spalte amount

- Nun wollen wir noch speichern, wie viele Einheiten der Kunde von dem Produkt bestellt hat.
- Wir erstellen also eine Spalte namens `amount` vom Datentyp `INT`.
- Die Menge muss immer angegeben werden, also definieren wir sie als `NOT NULL`.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT    NOT NULL REFERENCES customer(id),
7      product     INT    NOT NULL REFERENCES product(id),
8      amount      INT    NOT NULL,
9      ordered     DATE NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Die Spalte ordered

- Wir müssen auch das Bestelldatum speichern.



Die Spalte ordered

- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).



Die Spalte ordered

- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?



Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?
- Zum Glück gibt es in SQL Datentypen für Datums und Zeitangaben.

Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?
- Zum Glück gibt es in SQL Datentypen für Datums und Zeitangaben.
- Hier verwenden wir den Datentyp `DATE`.

Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?
- Zum Glück gibt es in SQL Datentypen für Datums und Zeitangaben.
- Hier verwenden wir den Datentyp `DATE`, der ein Datum nach unserem (dem Gregorianischen) Kalender darstellt^{27,39}.

Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?
- Zum Glück gibt es in SQL Datentypen für Datums und Zeitangaben.
- Hier verwenden wir den Datentyp `DATE`, der ein Datum nach unserem (dem Gregorianischen) Kalender darstellt^{27,39}.
- Also Schreibweise wird das ISO-Standardformat „`YYYY-MM-DD`“¹⁶ verwendet, wobei `YYYY` das Jahr in vier Ziffern, `MM` den Monat in zwei Ziffern und `DD` den Tag in zwei Ziffern darstellen.

Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered` (da `WHEN` und `DATE` reservierte SQL-Schlüsselworte sind⁴⁹).
- Aber welchen Datentyp sollten wir benutzen?
- Zum Glück gibt es in SQL Datentypen für Datums und Zeitangaben.
- Hier verwenden wir den Datentyp `DATE`, der ein Datum nach unserem (dem Gregorianischen) Kalender darstellt^{27,39}.
- Also Schreibweise wird das ISO-Standardformat „`YYYY-MM-DD`“¹⁶ verwendet, wobei `YYYY` das Jahr in vier Ziffern, `MM` den Monat in zwei Ziffern und `DD` den Tag in zwei Ziffern darstellen.
- Als Sicherheitscheck definieren wir ein `CONSTRAINT`, dass sicherstellt das nur Bestellungen nach dem 01.10.2024 und vor dem 01.01.3000 eingegeben werden können.

Die Spalte ordered



- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered`.
- Hier verwenden wir den Datentyp `DATE`.

```
1  /* We create the new table 'demand' in our factory database. */
2
3  -- The table 'demand' stores all the customer orders.
4  CREATE TABLE demand (
5      id          INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
6      customer    INT    NOT NULL REFERENCES customer(id),
7      product     INT    NOT NULL REFERENCES product(id),
8      amount      INT    NOT NULL,
9      ordered     DATE   NOT NULL,
10     CONSTRAINT ordered_amount_ok CHECK (
11         (amount > 0) AND (amount < 1000000)),
12     CONSTRAINT ordered_date_ok CHECK (
13         (ordered > '2024-10-01') AND (ordered < '3000-01-01'))
14 );
15
16 -- List all tables of the user 'boss' in database 'factory'
17 -- Now we see the table 'demand'.
18 SELECT tablename FROM pg_catalog.pg_tables
19     WHERE tableowner='boss';
```

Die Spalte ordered

- Wir müssen auch das Bestelldatum speichern.
- Wir erstellen also eine Spalte namens `ordered`.
- Hier verwenden wir den Datentyp `DATE`.
- Damit ist unsere Tabelle `demand` fertig.





Einfügen von Daten



Daten Einfügen

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



Daten Einfügen



- Nun haben wir die Tabelle `demand` erstellt, aber sie ist leer.
- Wir wollen nun folgende acht Bestellungen speichern:

Daten Einfügen



- Nun haben wir die Tabelle `demand` erstellt, aber sie ist leer.
- Wir wollen nun folgende acht Bestellungen speichern:

id	customer	product	amount	ordered
1	1 (Bibbo)	7 (Shoe, Size 42)	12	2024-11-21
2	2 (Bebbo)	3 (Shoe, Size 38)	2	2024-12-09
3	3 (Bebba)	2 (Shoe, Size 37)	7	2024-12-16
4	2 (Bebbo)	5 (Shoe, Size 40)	7	2024-12-30
5	1 (Bibbo)	5 (Shoe, Size 40)	3	2025-01-05
6	2 (Bebbo)	6 (Shoe, Size 41)	4	2025-01-12
7	3 (Bebba)	11 (Large Purse)	10	2025-01-16
8	2 (Bebbo)	3 (Shoe, Size 38)	6	2025-02-05
...	...	...	...	...

Füllen der Tabelle demand



```
1  /* Store some data into the table 'demand'. */
2
3  -- Print all the contents from table 'demand': Nothing.
4  SELECT * FROM demand;
5
6  -- Insert 8 orders into our table.
7  INSERT INTO demand (customer, product, amount, ordered)
8  VALUES (1, 7, 12, '2024-11-21'), (2, 3, 2, '2024-12-09'),
9          (3, 2, 7, '2024-12-16'), (2, 5, 7, '2024-12-30'),
10         (1, 5, 3, '2025-01-05'), (2, 6, 4, '2025-01-12'),
11         (3, 11, 10, '2025-01-16'), (2, 3, 6, '2025-02-05');
12
13  -- Now there are 8 rows.
14  SELECT * FROM demand;
```

Füllen der Tabelle demand



```
1 $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
  ↳ =1 -ebf insert_into_table_demand.sql
2 id | customer | product | amount | ordered
3 ---+-----+-----+-----+-----
4 (0 rows)
5
6 INSERT 0 8
7 id | customer | product | amount | ordered
8 ---+-----+-----+-----+-----
9 1 | 1 | 7 | 12 | 2024-11-21
10 2 | 2 | 3 | 2 | 2024-12-09
11 3 | 3 | 2 | 7 | 2024-12-16
12 4 | 2 | 5 | 7 | 2024-12-30
13 5 | 1 | 5 | 3 | 2025-01-05
14 6 | 2 | 6 | 4 | 2025-01-12
15 7 | 3 | 11 | 10 | 2025-01-16
16 8 | 2 | 3 | 6 | 2025-02-05
17 (8 rows)
18
19 # psql 16.11 succeeded with exit code 0.
```

Schutz vor Fehlern

- Schützt das DBMS uns auch wirklich vor Fehlern?



Schutz vor Fehlern



- Schützt das DBMS uns auch wirklich vor Fehlern?
- Was passiert, wenn wir eine falsche Produkt-ID verwenden?

```
1  /* Store some incorrect data into the table 'demand'. */  
2  
3  -- Trying to insert an order with an invalid product ID.  
4  INSERT INTO demand (customer, product, amount, ordered)  
5  VALUES (1, 77, 12, '2024-11-21');
```

Schutz vor Fehlern



- Schützt das DBMS uns auch wirklich vor Fehlern?
- Wir können keine falsche Produkt-ID vwerden.

```
1  /* Store some incorrect data into the table 'demand'. */
2
3  -- Trying to insert an order with an invalid product ID.
4  INSERT INTO demand (customer, product, amount, ordered)
5  VALUES (1, 77, 12, '2024-11-21');
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↳ =1 -ebf insert_into_table_demand_error_1.sql
2  psql:factory/insert_into_table_demand_error_1.sql:5: ERROR:  insert or
   ↳ update on table "demand" violates foreign key constraint "
   ↳ demand_product_fkey"
3  DETAIL:  Key (product)=(77) is not present in table "product".
4  psql:factory/insert_into_table_demand_error_1.sql:5: STATEMENT:  /*
   ↳ Store some incorrect data into the table 'demand'. */
5  -- Trying to insert an order with an invalid product ID.
6  INSERT INTO demand (customer, product, amount, ordered)
7  VALUES (1, 77, 12, '2024-11-21');
8  # psql 16.11 failed with exit code 3.
```

Schutz vor Fehlern



- Schützt das DBMS uns auch wirklich vor Fehlern?
- Wir können keine falsche Produkt-ID vwerden.
- Was passiert, wenn wir ein falsches Datum eingeben?

```
1  /* Store some incorrect data into the table 'demand'. */  
2  
3  -- Trying to insert an order with an invalid date.  
4  INSERT INTO demand (customer, product, amount, ordered)  
5  VALUES (1, 7, 12, '1024-11-21');
```

Schutz vor Fehlern



- Schützt das DBMS uns auch wirklich vor Fehlern?
- Wir können keine falsche Produkt-ID vwerden.
- Wir können kein ungültiges Datum eingeben.

```
1  /* Store some incorrect data into the table 'demand'. */
2
3  -- Trying to insert an order with an invalid date.
4  INSERT INTO demand (customer, product, amount, ordered)
5  VALUES (1, 7, 12, '1024-11-21');
```

```
1  $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf insert_into_table_demand_error_2.sql
2  psql:factory/insert_into_table_demand_error_2.sql:5: ERROR:  new row for
   ↪ relation "demand" violates check constraint "ordered_date_ok"
3  DETAIL:  Failing row contains (10, 1, 7, 12, 1024-11-21).
4  psql:factory/insert_into_table_demand_error_2.sql:5: STATEMENT:  /*
   ↪ Store some incorrect data into the table 'demand'. */
5  -- Trying to insert an order with an invalid date.
6  INSERT INTO demand (customer, product, amount, ordered)
7  VALUES (1, 7, 12, '1024-11-21');
8  # psql 16.11 failed with exit code 3.
```

Understanding the Data



- Wenn wir uns die Daten in unserer skizzierten Tabelle anschauen, dann können wir sie ganz gut verstehen, weil wir die Bedeutungen der IDs mit rangeschrieben haben.

id	customer	product	amount	ordered
1	1 (Bibbo)	7 (Shoe, Size 42)	12	2024-11-21
2	2 (Bebbo)	3 (Shoe, Size 38)	2	2024-12-09
3	3 (Bebba)	2 (Shoe, Size 37)	7	2024-12-16
4	2 (Bebbo)	5 (Shoe, Size 40)	7	2024-12-30
5	1 (Bibbo)	5 (Shoe, Size 40)	3	2025-01-05
6	2 (Bebbo)	6 (Shoe, Size 41)	4	2025-01-12
7	3 (Bebba)	11 (Large Purse)	10	2025-01-16
8	2 (Bebbo)	3 (Shoe, Size 38)	6	2025-02-05
...	...	...	...	...



Understanding the Data

- Wenn wir uns die Daten in unserer skizzierten Tabelle anschauen, dann können wir sie ganz gut verstehen, weil wir die Bedeutungen der IDs mit rangeschrieben haben.
- Ein **SELECT** gibt uns aber nur die IDs für Produkte und Kunden ... und ist daher schwer zu verstehen

```
1 $ psql "postgres://boss:superboss123@localhost/factory" -v ON_ERROR_STOP
   ↪ =1 -ebf insert_into_table_demand.sql
2 id | customer | product | amount | ordered
3 -----
4 (0 rows)
5
6 INSERT 0 8
7 id | customer | product | amount | ordered
8 -----
9 1 | 1 | 7 | 12 | 2024-11-21
10 2 | 2 | 3 | 2 | 2024-12-09
11 3 | 3 | 2 | 7 | 2024-12-16
12 4 | 2 | 5 | 7 | 2024-12-30
13 5 | 1 | 5 | 3 | 2025-01-05
14 6 | 2 | 6 | 4 | 2025-01-12
15 7 | 3 | 11 | 10 | 2025-01-16
16 8 | 2 | 3 | 6 | 2025-02-05
17 (8 rows)
18
19 # psql 16.11 succeeded with exit code 0.
```

Understanding the Data



- Wenn wir uns die Daten in unserer skizzierten Tabelle anschauen, dann können wir sie ganz gut verstehen, weil wir die Bedeutungen der IDs mit rangeschrieben haben.
- Ein `SELECT` gibt uns aber nur die IDs für Produkte und Kunden ... und ist daher schwer zu verstehen.
- In der nächsten Einheit lernen wir, wie wir die Daten aus den verschiedenen Tabellen zusammenführen und auswerten können.



Zusammenfassung



Zusammenfassung



- Nun haben wir die dritte und letzte Tabelle unserer Fabrikdatenbank erstellt.

Zusammenfassung



- Nun haben wir die dritte und letzte Tabelle unserer Fabrikdatenbank erstellt.
- Anders als die beiden Tabellen davor existiert sie nicht „alleine für sich.“

- Nun haben wir die dritte und letzte Tabelle unserer Fabrikdatenbank erstellt.
- Anders als die beiden Tabellen davor existiert sie nicht „alleine für sich.“
- Stattdessen referenziert sie die Tabellen `customer` und `product` über Fremdschlüssel, also in dem sie Spalten hat, die Werte von deren Primärschlüssel speichern.

- Nun haben wir die dritte und letzte Tabelle unserer Fabrikdatenbank erstellt.
- Anders als die beiden Tabellen davor existiert sie nicht „alleine für sich.“
- Stattdessen referenziert sie die Tabellen `customer` und `product` über Fremdschlüssel, also in dem sie Spalten hat, die Werte von deren Primärschlüssel speichern.
- Somit können wir speichern, welcher Kunde welches Produkt bestellt hat.

- Nun haben wir die dritte und letzte Tabelle unserer Fabrikdatenbank erstellt.
- Anders als die beiden Tabellen davor existiert sie nicht „alleine für sich.“
- Stattdessen referenziert sie die Tabellen `customer` und `product` über Fremdschlüssel, also in dem sie Spalten hat, die Werte von deren Primärschlüssel speichern.
- Somit können wir speichern, welcher Kunde welches Produkt bestellt hat.
- Als nächstes werden wir lernen, wie wir die Daten in unserer Datenbank auswerten können.



谢谢你们！
Thank you!
Vielen Dank!



References I



- [1] 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 also² (siehe S. 98, 104).
- [2] 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 also¹ (siehe S. 98, 104).
- [3] 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. 104, 106).
- [4] 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. 104).
- [5] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) and 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. 106).
- [6] 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. 104).
- [7] 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. 105).
- [8] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 105).
- [9] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 104).
- [10] 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. 13).
- [11] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 104, 105).

References II



- [12] 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](https://doi.org/10.1145/3649887). URL: <https://cacm.acm.org/research/50-years-of-queries> (besucht am 2025-01-09) (siehe S. 105).
- [13] 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. 106).
- [14] 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](https://doi.org/10.1145/362384.362685). URL: <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf> (besucht am 2025-01-05) (siehe S. 105).
- [15] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 105).
- [16] *Date and Time – Representations for Information Interchange – Part 1: Basic Rules*. International Standard ISO 8601-1:2019(E), Edition 1. Geneva, Switzerland: International Organization for Standardization (ISO), Feb. 2019 (siehe S. 67–74).
- [17] 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. 105).
- [18] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 105).
- [19] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebWJ: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: **978-0-13-299045-5** (siehe S. 106).
- [20] 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. 104, 105).
- [21] 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. 105).

References III



- [22] "Foreign Keys". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 5.5.5. URL: <https://www.postgresql.org/docs/17/ddl-constraints.html#DDL-CONSTRAINTS-FK> (besucht am 2025-02-28) (siehe S. 48–58).
- [23] 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. 105).
- [24] 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. 105).
- [25] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 104).
- [26] 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. 104, 106).
- [27] "History of Units". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. B.6. URL: <https://www.postgresql.org/docs/17/datetime-units-history.html> (besucht am 2025-03-01) (siehe S. 72–74).
- [28] 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. 105).
- [29] *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. 105).
- [30] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 105).
- [31] 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. 105).

References IV



- [32] 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. 104).
- [33] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 105).
- [34] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: <https://mariadb.com/kb/en/documentation> (besucht am 2025-04-24) (siehe S. 104).
- [35] 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. 105).
- [36] 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. 104).
- [37] 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. 105).
- [38] 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. 104).
- [39] Pope Gregory XIII. *Inter Gravissimas*. Proclamation / Papal Bull. Vatican: Catholic Church, 24. Feb. 1582 (siehe S. 72–74).
- [40] *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).
- [41] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 105).
- [42] 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. 104).

References V



- [43] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 104).
- [44] 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. 105).
- [45] 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. 104).
- [46] 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. 104).
- [47] 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. 105).
- [48] "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. 105).
- [49] "SQL Key Words". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Appendix C. URL: <https://www.postgresql.org/docs/17/sql-keywords-appendix.html> (besucht am 2025-07-04) (siehe S. 67–74).
- [50] 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. 102, 105).
- [51] 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. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of⁵⁰ (siehe S. 105).

References VI



- [52] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 105).
- [53] 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. 105).
- [54] 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. 104).
- [55] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 104).
- [56] 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. 104, 105).
- [57] 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. 105).
- [58] *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. 105).
- [59] 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. 105).
- [60] 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. 104).
- [61] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 104).

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^{9,36,61}. 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^{6,32,38,42,45}. 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*⁵⁶.

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 DB⁶⁰.

IT information technology

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP^{11,26}.

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows^{3,25,46,54,55}. 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^{1,2,4,20,34,43}. See <https://mariadb.org> for more information.

Glossary (in English) II



- Microsoft Windows is a commercial proprietary operating system⁸. 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>.
- MySQL An open source relational database management system^{7,20,44,53,59}. MySQL is famous for its use in the LAMP Stack. See <https://www.mysql.com> for more information.
- PostgreSQL An open source object-relational DBMS^{21,37,41,53}. See <https://postgresql.org> for more information.
- psql is the client program used to access the PostgreSQL DBMS server.
- Python The Python programming language^{28,31,33,57}, i.e., what you will learn about in our book⁵⁷. 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^{14,23,24,47,52,56,58}.
- 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 servers¹¹ 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“³⁰.
- SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases^{12,15,17,18,29,35,48,50–52}. It is understood by many DBMSes. You find the SQL commands supported by PostgreSQL in the reference⁴⁸.

Glossary (in English) III



terminal A terminal is a text-based window where you can enter commands and execute them^{3,13}. 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 Drück auf  + **R**, dann Schreiben von `cmd`, dann Drück auf . Under Ubuntu Linux, **Ctrl** + **Alt** + **T** opens a terminal, which then runs a Bash shell inside.

Ubuntu is a variant of the open source operating system Linux^{13,26}. 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>.

WWW World Wide Web^{5,19}