Session 9 – Normalisierung & ER-Diagramme

> **Interaktive Übung:** Gemeinsam entwickeln wir normalisierte Schemas und ER-Diagramme – von chaotisch zu strukturiert.

Überblick

Willkommen zur interaktiven Normalisierungs-Session! Heute arbeiten wir **zusammen** – ihr seid nicht Zuschauer, sondern Co-Entwickler. Wir durchlaufen:

0. **ER-Diagramme**: Was können wir damit beschreiben? Wie modellieren wir Entitäten und Beziehungen?
1. **Normalisierung (1NF → 2NF → 3NF)** am Beispiel eines Online-Shops
2. **Praxis-Übung**: Schritt für Schritt bauen wir Twitter

Lernziele:

- ER-Diagramme lesen und erstellen
- Normalisierungsschritte verstehen und anwenden (inkl. Transitivität)
- Komplexe Datenmodelle gemeinsam entwickeln

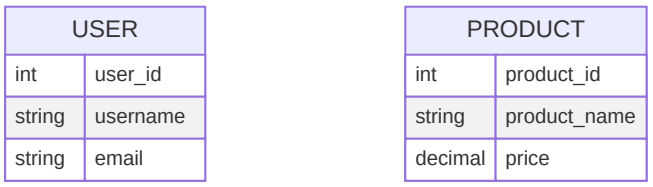
Teil 0: ER-Diagramme – Entitäten & Beziehungen

ER-Diagramme (Entity-Relationship-Diagramme) sind visuelle Werkzeuge zur Datenmodellierung. Sie zeigen Entitäten (Objekte), ihre Attribute und die Beziehungen zwischen ihnen.

Entitäten:

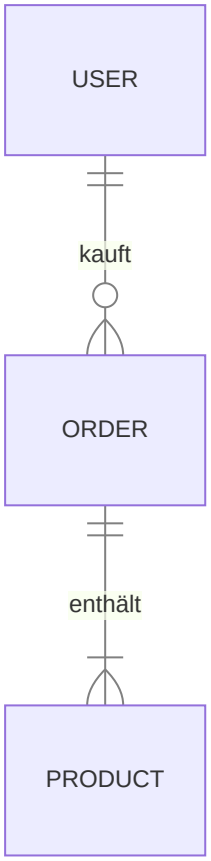
- Rechtecke (z.B. `User`, `Product`)

- Attribute als Ovale oder in der Entität

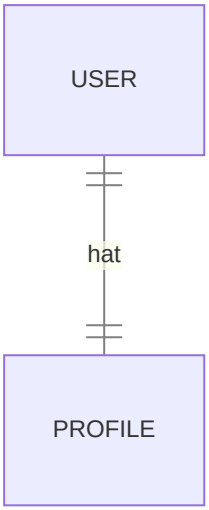


Beziehungen:

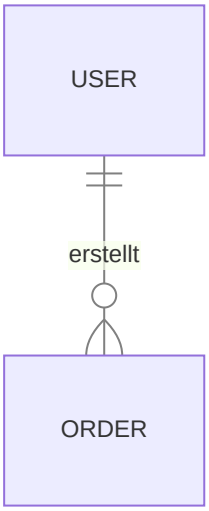
- Rauten oder Linien (z.B. `kauft`, `gehört zu`)



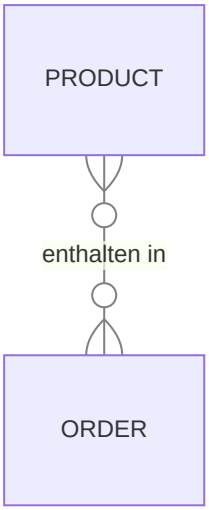
- Kardinalitäten: 1:1, 1:N, N:M
  - **1:1 (Eins-zu-Eins):** Ein User hat genau ein Profil, ein Profil gehört zu genau einem User



  - **1:N (Eins-zu-Viele):** Ein User kann viele Orders haben, eine Order gehört zu einem User



- **N:M (Viele-zu-Viele):** Viele Users können viele Products kaufen, viele Products können von vielen Users gekauft werden



Schlüssel:

- Primärschlüssel (unterstrichen)

USER		
int	user_id	PK
string	username	
string	email	

- Fremdschlüssel (gestrichelt)

ORDER		
int	order_id	PK
int	user_id	FK
date	order_date	

Teil 1: Normalisierung – Von Chaos zu Struktur

Problem: Der chaotische Online-Shop

Stellt euch vor: Ihr habt gerade einen Online-Shop geerbt. Alle Daten liegen in einer riesigen Tabelle. Klingt praktisch? Schauen wir mal...

Schritt 0: Die Ausgangslage (0NF)

Hier ist unsere „All-in-One“ Tabelle. Auf den ersten Blick funktioniert sie – aber schaut genauer hin: Was fällt euch auf?

Die chaotische Tabelle

```sql
-- Erstelle die chaotische "Alles-in-Einem" Tabelle
CREATE TABLE shop_nf0 (
    order_id INTEGER,
    order_date DATE,
    customer_id INTEGER,
    customer_name VARCHAR(100),
    email VARCHAR(100),
    address VARCHAR(100),
    product_id INTEGER,
    product_name VARCHAR(100),
    price DECIMAL(10,2),
    category VARCHAR(50),
    quantity INTEGER
);

-- Befülle mit realistischen Beispieldaten (mit nicht-atomaren Adress
INSERT INTO shop_nf0 VALUES
(1, '2024-01-15', 1, 'Anna Müller', 'anna@mail.de', 'Hauptstraße 12,
  Leipzig', 1, 'Laptop Dell XPS', 1299.99, 'Elektronik', 1),
(2, '2024-01-16', 1, 'Anna Müller', 'anna@mail.de', 'Hauptstraße 12,
  Leipzig', 1, 'USB-C Kabel', 15.99, 'Zubehör', 2),
(3, '2024-01-16', 2, 'Max Schmidt', 'max.s@web.de', 'Berliner Allee 4
  10115 Berlin', 1, 'Laptop Dell XPS', 1299.99, 'Elektronik', 1),
(4, '2024-01-17', 3, 'Lisa Weber', 'lisa.w@gmail.com', 'Hafenstraße 8
  20095 Hamburg', 1, 'Wireless Maus', 29.99, 'Zubehör', 1);

-- Schauen wir uns das Chaos an
SELECT * FROM shop_nf0;
```

```
-- Erstelle die chaotische "Alles-in-Einem" Tabelle
CREATE TABLE shop_nf0 (
    order_id INTEGER,
    order_date DATE,
    customer_id INTEGER,
    customer_name VARCHAR(100),
    email VARCHAR(100),
    address VARCHAR(100),
    product_id INTEGER,
    product_name VARCHAR(100),
    price DECIMAL(10,2),
    category VARCHAR(50),
    quantity INTEGER
)
```

ok

```
-- Befülle mit realistischen Beispieldaten (mit nicht-atomaren Adressen!)
INSERT INTO shop_nf0 VALUES
(1, '2024-01-15', 1, 'Anna Müller', 'anna@mail.de', 'Hauptstraße 12, 04109 Leipzig', 1,
'Laptop Dell XPS', 1299.99, 'Elektronik', 1),
(2, '2024-01-16', 1, 'Anna Müller', 'anna@mail.de', 'Hauptstraße 12, 04109 Leipzig', 1,
'USB-C Kabel', 15.99, 'Zubehör', 2),
(3, '2024-01-16', 2, 'Max Schmidt', 'max.s@web.de', 'Berliner Allee 45, 10115 Berlin',
1, 'Laptop Dell XPS', 1299.99, 'Elektronik', 1),
(4, '2024-01-17', 3, 'Lisa Weber', 'lisa.w@gmail.com', 'Hafenstraße 8, 20095
Hamburg', 1, 'Wireless Maus', 29.99, 'Zubehör', 1)
```

ok

```
-- Schauen wir uns das Chaos an
SELECT * FROM shop_nf0
```

#	order_id	order_date	customer_id	customer_name	email	address
1	1	2024-01-15	1	Anna Müller	anna@mail.de	Hauptstraße 12, 04109 Leipzig
2	2	2024-01-16	1	Anna Müller	anna@mail.de	Hauptstraße 12, 04109 Leipzig
3	3	2024-01-16	2	Max Schmidt	max.s@web.de	Berliner Allee 45, 10115 Berlin
4	4	2024-01-17	3	Lisa Weber	lisa.w@gmail.com	Hafenstraße 8, 20095 Hamburg

4 rows

Gibt es Probleme, die man direkt sehen kann?

Schritt 1: Erste Normalform (1NF)

Regel: Eine Tabelle ist in 1NF, wenn alle Attribute atomar sind (keine Listen, keine verschachtelten Strukturen). Jede Zelle enthält genau einen Wert.

Was ändert sich?

Aufgabe: Identifiziert Spalten, die gegen 1NF verstoßen.

```
1  CREATE TABLE shop_nf1 (
2
3  );
4
5  --INSERT INTO shop_nf1 VALUES
6
7  --SELECT * FROM shop_nf1;
```

```
CREATE TABLE shop_nf1 (

)
```

ok

```
--INSERT INTO shop_nf1 VALUES

--SELECT * FROM shop_nf1;
```

ok

Schritt 2: Zweite Normalform (2NF)

Regel: Eine Tabelle ist in 2NF, wenn sie in 1NF ist und jedes Nicht-Schlüssel-Attribut voll funktional vom Primärschlüssel abhängt (keine partiellen Abhängigkeiten).

Was bedeutet das konkret?

```
1  -- TODO: Zerlege die Tabelle in 2NF-konforme Tabellen
2
3
```

```
-- TODO: Zerlege die Tabelle in 2NF-konforme Tabellen
```

ok

Frage an euch:

- Welche Daten hängen nur von einem Teil des Schlüssels ab?
- Welche Tabellen fehlen uns noch?

Schritt 3: Dritte Normalform (3NF) – Transitivität eliminieren

Regel: Eine Tabelle ist in 3NF, wenn sie in 2NF ist und kein Nicht-Schlüssel-Attribut transitiv vom Primärschlüssel abhängt.

Was sind transitive Abhängigkeiten?

```
-- TODO: Zerlege die Tabelle in 3NF-konforme Tabellen


```

```
-- TODO: Zerlege die Tabelle in 3NF-konforme Tabellen
```

ok

- Welche Tabelle brauchen wir zusätzlich?
- Wann ist das praktikabel, wann übertrieben?

Finales normalisiertes Schema

Das ist unser Ziel: Ein sauberes, normalisiertes Schema mit klaren Beziehungen und ohne Redundanz.

Unser Online-Shop (0NF)

dbdiagram.io

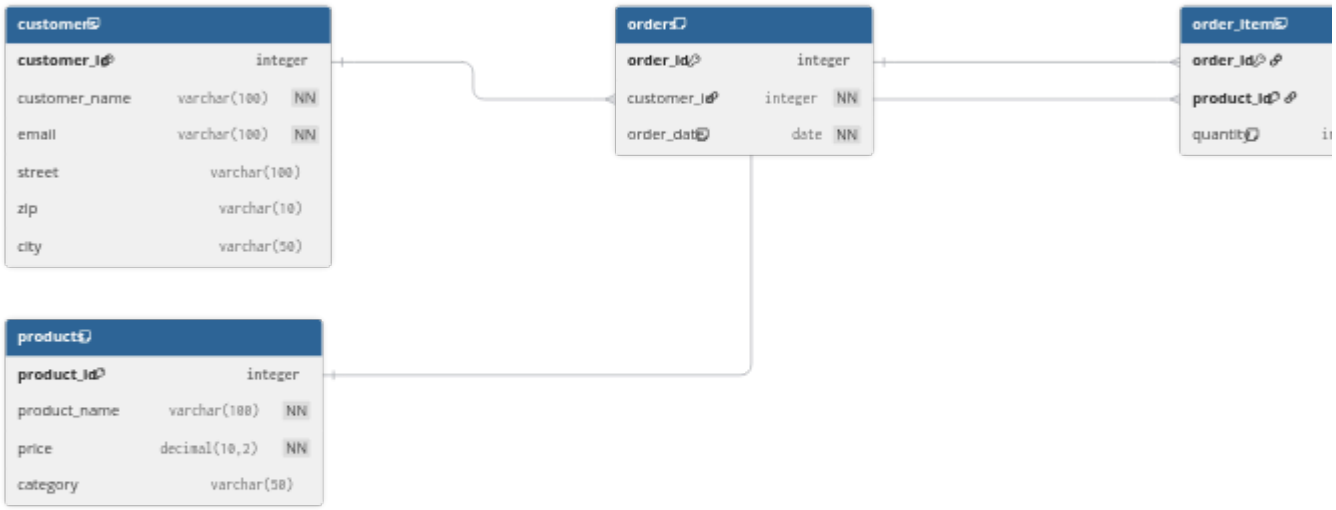
dbdiagram.io

Unser Online-Shop (1NF)

dbdiagram.io

dbdiagram.io

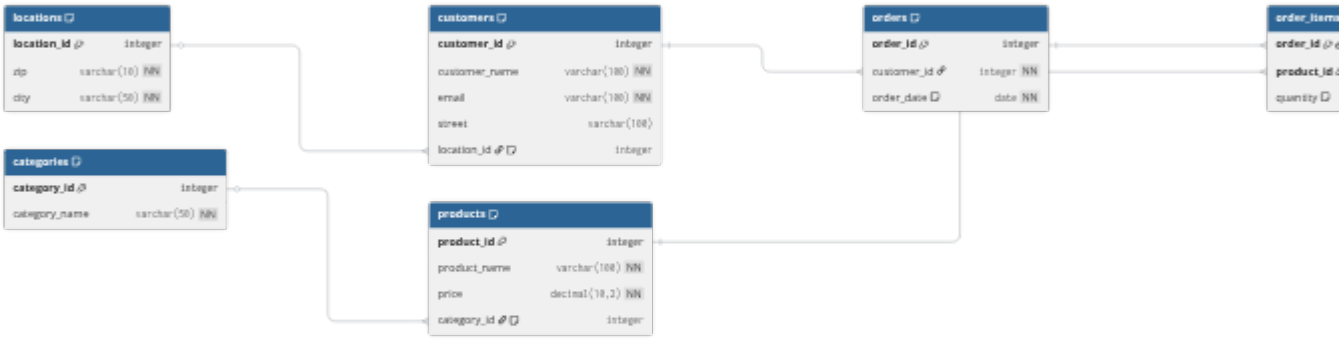
Unser Online-Shop (2NF)



dbdiagram.io

[dbdiagram.io]

Unser Online-Shop (3NF)



dbdiagram.io

[dbdiagram.io]

Diskussion: - Wann würdet ihr denormalisieren? - Performance vs. Konsistenz?

Teil 2: Twitter-Modell gemeinsam entwickeln

> Aufgabe: Baue twitter nach und befülle es mit Beispieldaten – Schritt für Schritt!

Twitter.db

```
1
2
3
4
5
6
7
8
9
10
11
```