DATA DRIVEN DECISIONS

Sommersemester 2026

Datenmanagement & -analyse

Prof. Dr. Christoph M. Flath
Lehrstuhl für Wirtschaftsinformatik und Business Analytics

Julius-Maximilians-Universität Würzburg

DATA DRIVEN DECISIONS

Rückblick: Der Modellierungsprozess

Letzte Session:

- Entitäten, Attribute, Beziehungen
- Kardinalitäten (1:1, 1:N, M:N)
- Chen-Notation

Diese Session:

- ER-Modell → Tabellen (Relationales Schema)
- CREATE TABLE mit Schlüsseln

Das Relationale Modell

Erfinder: Edgar F. Codd (IBM, 1970)

Kernidee: Alle Daten in **Tabellen** (Relationen)

Terminologie:

- **Relation** = Tabelle
- **Tupel** = Zeile (Datensatz)
- **Attribut** = Spalte
- **Domäne** = Wertebereich einer Spalte

Beispiel:
Relation `Spieler` mit 10 Tupeln und 4 Attributen

ID	Name	Pos	Alter
1	Müller	Sturm	35
2	Neuer	Tor	38
3	Kimmich	Mittelfeld	29
...			

↑ Attribute (Spalten)
← Tupel (Zeilen)

Eigenschaften einer Relation

DATA DRIVEN DECISIONS

Regeln für Relationen:

1. Jede Zeile ist **eindeutig**
2. Jede Spalte hat einen **eindeutigen Namen**
3. Alle Werte einer Spalte haben **gleichen Typ**
4. Werte sind **atomar** (nicht teilbar)
5. **Reihenfolge** der Zeilen/Spalten ist egal

Relationales Schema:

Beschreibung der Tabellenstruktur:

Spieler(ID, Name, Position, Alter)

- Tabellenname
- Attribute in Klammern
- Primärschlüssel unterstrichen

Notation

Tabellenname(PK, Attribut1, Attribut2, #FK)

- Unterstrichen = Primärschlüssel
- # = Fremdschlüssel

DATA DRIVEN DECISIONS

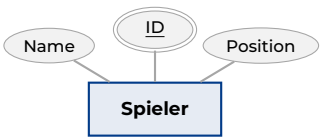
Warum diese Transformation?

ER-Modell:

- Konzeptuelles Modell
- Für Menschen verständlich
- Graphische Notation
- Unabhängig vom DBMS

Relationales Schema:

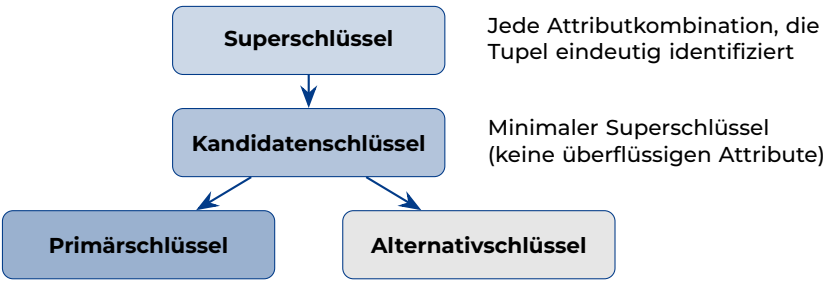
- Logisches Modell
- Für DBMS umsetzbar
- Textbasierte Notation
- Direkt in SQL übersetzbar



$\Rightarrow$ Spieler(<u>ID</u>, Name, Position)

Agenda

Schlüssel: Übersicht



Schlüssel: Beispiel

Tabelle Mitarbeiter:

MitarbeiterNr	SVNr	Name	Abteilung	Gehalt
101	12345678	Müller	IT	5000
102	23456789	Schmidt	HR	4500
103	34567890	Müller	IT	5500

Analyse:

- **Superschlüssel:** {MitarbeiterNr}, {SVNr}, {MitarbeiterNr, Name}, {MitarbeiterNr, SVNr}, ...
- **Kandidatenschlüssel:** {MitarbeiterNr}, {SVNr}
- **Kein Schlüssel:** {Name} (Duplikat: Müller), {Abteilung}, {Gehalt}

Entscheidung

Wir wählen **MitarbeiterNr** als Primärschlüssel → **SVNr** wird Alternativschlüssel

Primärschlüssel (Primary Key)

Definition:
Attribut(e), die jede Zeile **eindeutig identifizieren**.

Eigenschaften:

- **Eindeutig** – keine Duplikate
- **Nicht NULL** – immer ein Wert
- **Stabil** – ändert sich nicht
- **Minimal** – keine überflüssigen Teile

Natürlich vs. Künstlich:

- Natürlich: ISBN, Matrikelnummer
- Künstlich: Auto-ID (1, 2, 3, ...)

ID	Name	Position
1	Müller	Sturm
2	Neuer	Tor
3	Müller	Abwehr

"Name" wäre kein guter PK
(zwei Müller möglich!)

Best Practice

Im Zweifelsfall: Künstlicher Schlüssel (ID)

Zusammengesetzter Primärschlüssel

Definition: Primärschlüssel aus **mehreren Attributen**

Wann nötig?

- Kein einzelnes Attribut ist eindeutig
- Beziehungstabellen bei M:N-Beziehungen
- Schwache Entitäten

Beispiel: Buchung

Flug_ID	Sitz_Nr	Passagier	Datum
LH123	12A	Müller	2026-05-15
LH123	12B	Schmidt	2026-05-15
LH456	12A	Weber	2026-05-15

- Flug_ID allein nicht eindeutig (mehrere Sitze)
- Sitz_Nr allein nicht eindeutig (mehrere Flüge)
- **Kombination** (Flug_ID, Sitz_Nr) ist eindeutig!

Fremdschlüssel (Foreign Key)

Definition:
Attribut, das auf den **Primärschlüssel einer anderen Tabelle** verweist.

Zweck:

- Verbindet Tabellen
- Stellt Beziehungen her
- Erzwingt **referentielle Integrität**

Referentielle Integrität:
Ein Fremdschlüssel muss auf einen **existierenden** Primärschlüssel zeigen.

Spieler

ID	Name	#Verein_ID
1	Müller	1
2	Wirtz	2

↓ verweist auf ↓

Verein

ID	Name
1	Bayern München
2	Bayer Leverkusen

Referentielle Integrität: Warum wichtig?

DATA DRIVEN DECISIONS

Was passiert ohne Integritätsprüfung?

ID	Name	#Verein_ID
1	Müller	99

↓ Verein 99 existiert nicht! ↓

ID	Name
1	Bayern München
2	Bayer Leverkusen

Probleme:

- ✗ Daten sind inkonsistent
- ✗ JOINs liefern unvollständige Ergebnisse
- ✗ Anwendungen können abstürzen

Lösung

Das DBMS prüft automatisch: FK-Wert muss in referenzierter Tabelle existieren!

Fremdschlüssel: Was passiert bei Löschung?

Problem: Was passiert, wenn ein referenzierter Datensatz gelöscht wird?

Option	SQL	Beschreibung
Verbieten	`RESTRICT`	Löschung wird abgelehnt
Kaskadieren	`CASCADE`	Abhängige Zeilen werden mitgelöscht
NULL setzen	`SET NULL`	Fremdschlüssel wird auf NULL gesetzt
Default	`SET DEFAULT`	Fremdschlüssel wird auf Default gesetzt
Nichts tun	`NO ACTION`	Prüfung am Ende der Transaktion

Beispiel: Verein wird gelöscht

- `RESTRICT`: Fehler, wenn noch Spieler existieren
- `CASCADE`: Alle Spieler des Vereins werden gelöscht
- `SET NULL`: Spieler haben keinen Verein mehr (NULL)

Fremdschlüssel-Aktionen: Beispiele in SQL

DATA DRIVEN DECISIONS

```sql
-- RESTRICT: Verein kann nicht gelöscht werden,
--           wenn noch Spieler existieren
FOREIGN KEY (Verein_ID) REFERENCES Verein(ID)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT

-- CASCADE: Bei Löschung werden Spieler mitgelöscht
FOREIGN KEY (Verein_ID) REFERENCES Verein(ID)
    ON DELETE CASCADE
    ON UPDATE CASCADE

-- SET NULL: Spieler behalten, aber Verein auf NULL
FOREIGN KEY (Verein_ID) REFERENCES Verein(ID)
    ON DELETE SET NULL
    ON UPDATE CASCADE
```

Best Practice

`ON DELETE RESTRICT` ist sicher – lieber explizit löschen!
`ON UPDATE CASCADE` ist meist sinnvoll bei PK-Änderungen.

Quiz: Fremdschlüssel-Aktionen

DATA DRIVEN DECISIONS

Welche Aktion wählen Sie für diese Szenarien?

Szenario 1: Bestellung → Kunde
Wenn Kunde gelöscht wird, sollen Bestellungen...

- **A** ...mitgelöscht werden (CASCADE)
- **B** ...erhalten bleiben, Kunde auf NULL (SET NULL)
- **C** ...Löschung verhindern (RESTRICT)

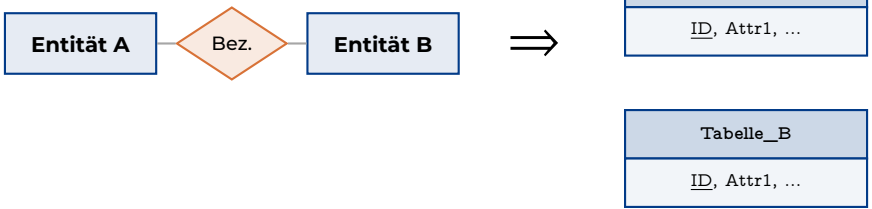
Szenario 2: Log-Eintrag → Benutzer
Wenn Benutzer gelöscht wird, sollen Logs...

- **A** ...mitgelöscht werden (CASCADE)
- **B** ...erhalten bleiben (SET NULL oder RESTRICT)

Diskutieren Sie!

DATA DRIVEN DECISIONS

Transformation: Grundprinzip

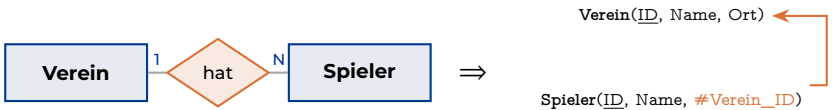


Grundregeln:

1. Jede **Entität** wird zu einer **Tabelle**
2. Jedes **Attribut** wird zu einer **Spalte**
3. Der **Schlüssel** wird zum **Primärschlüssel**
4. **Beziehungen** werden durch **Fremdschlüssel** oder **zusätzliche Tabellen** abgebildet

Transformation: 1:N-Beziehung

Regel: Fremdschlüssel auf der “N”-Seite



Warum auf der N-Seite?

- Ein Spieler gehört zu **einem** Verein → ein FK-Wert reicht
- Ein Verein hat **viele** Spieler → Liste wäre nötig (nicht atomar!)

Merksatz

Der Fremdschlüssel kommt auf die “N”-Seite (die Seite mit vielen).

1:N-Transformation: Schritt für Schritt

1. **Entitäten identifizieren:**
   Verein (1-Seite), Spieler (N-Seite)
2. **Tabellen erstellen:**
   Für jede Entität eine Tabelle mit allen Attributen
3. **Primärschlüssel festlegen:**
   Verein: ID, Spieler: ID
4. **Fremdschlüssel hinzufügen:**
   In Spieler: Verein_ID (zeigt auf Verein.ID)
5. **Beziehungsattribute zuordnen:**
   Falls vorhanden: zur N-Seite (Spieler)

Schritt	Ergebnis
Nach Schritt 2	Verein(ID, Name, Ort), Spieler(ID, Name)
Nach Schritt 4	Spieler(ID, Name, Verein_ID)

Beispiel: 1:N in SQL

ER-Modell:
Verein (1) ← hat → Spieler (N)

Relationales Schema:

- Verein(ID, Name, Ort)
- Spieler(ID, Name, #Verein_ID)

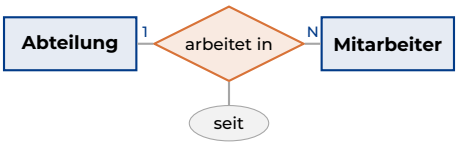
SQL:

```
CREATE TABLE Verein (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Ort VARCHAR(50)
);

CREATE TABLE Spieler (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Verein_ID INT,
    FOREIGN KEY (Verein_ID)
        REFERENCES Verein(ID)
);
```

1:N mit Beziehungsattribut

Frage: Was, wenn die Beziehung selbst Attribute hat?



Lösung: Beziehungsattribut kommt zur N-Seite!

Mitarbeiter(ID, Name, #Abt_ID, **seit**)

Warum zur N-Seite?

Das Attribut "seit" beschreibt, **seit wann** ein Mitarbeiter in der Abteilung arbeitet – es gehört zum Mitarbeiter, nicht zur Abteilung.

Wie transformiert man diese 1:N-Beziehung?

Optionen:

A Autor(ID, Name, #Buch_ID)

B Buch(ID, Titel, #Autor_ID)

C Neue Tabelle: Autor_Buch(#Autor_ID, #Buch_ID)

Wie transformiert man diese 1:N-Beziehung?

Optionen:

- **A** Autor(ID, Name, #Buch_ID)
- **B** Buch(ID, Titel, #Autor_ID)
- **C** Neue Tabelle: Autor_Buch(#Autor_ID, #Buch_ID)

Antwort: B – FK auf der N-Seite (Buch)!

Hands-on

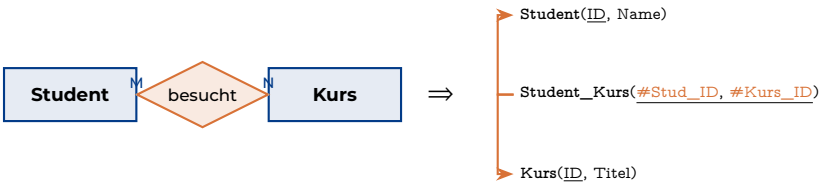
1:N-Beziehungen transformieren

marimo: 07-er-zu-sql.py

Aufgabe 7.1

Transformation: M:N-Beziehung

Regel: Zusätzliche **Beziehungstabelle** (Junction Table)



Die Beziehungstabelle:

- Enthält **zwei Fremdschlüssel** (zu beiden Entitäten)
- Der **Primärschlüssel** ist die Kombination beider FKs
- Kann **eigene Attribute** haben (z.B. Note, Datum)

M:N-Transformation: Schritt für Schritt

1. **Beide Entitäten als Tabellen:**
   Student(ID, Name), Kurs(ID, Titel)
2. **Beziehungstabelle erstellen:**
   Name: oft Kombination beider Entitäten (Student_Kurs)
3. **Fremdschlüssel hinzufügen:**
   Student_ID → Student.ID
   Kurs_ID → Kurs.ID
4. **Primärschlüssel festlegen:**
   PK = (Student_ID, Kurs_ID) – zusammengesetzt!
5. **Beziehungsattribute hinzufügen:**
   Falls vorhanden: Note, Einschreibedatum, etc.

Wichtig

Die Beziehungstabelle hat **keinen eigenen ID-Schlüssel** – die Kombination der FKs reicht!

Beispiel: M:N in SQL

DATA DRIVEN DECISIONS

```sql
-- Entitätstabellen
CREATE TABLE Student (
    ID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Kurs (
    ID INT PRIMARY KEY,
    Titel VARCHAR(200)
);

-- Beziehungstabelle
CREATE TABLE Student_Kurs (
    Student_ID INT,
    Kurs_ID INT,
    Note DECIMAL(2,1),          -- Beziehungsattribut!
    PRIMARY KEY (Student_ID, Kurs_ID),
    FOREIGN KEY (Student_ID) REFERENCES Student(ID),
    FOREIGN KEY (Kurs_ID) REFERENCES Kurs(ID)
);
```

M:N-Beziehungstabelle: Beispieldaten

Student

ID	Name
1	Anna
2	Ben
3	Clara

Kurs

ID	Titel
101	DMA
102	BWL

Student_Kurs

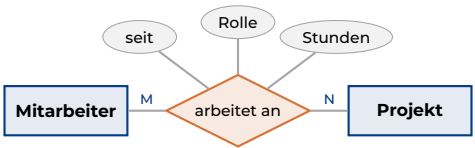
#Stud	#Kurs	Note
1	101	1.3
1	102	2.0
2	101	1.7
3	101	1.0
3	102	1.3

Lesebeispiel:

- Anna (1) besucht DMA (101) und BWL (102)
- Ben (2) besucht nur DMA (101)
- DMA wird von Anna, Ben und Clara besucht

M:N mit mehreren Beziehungsattributen

Beispiel: Projekt-Mitarbeiter-Zuordnung

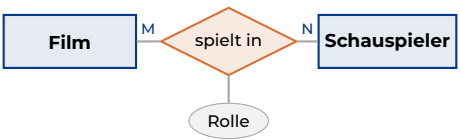


Beziehungstabelle:

Mitarbeiter_Projekt(#MA_ID, #Proj_ID, Rolle, Stunden, seit)

#MA_ID	#Proj_ID	Rolle	Stunden	seit
101	1	Leiter	20	2025-01-01
101	2	Berater	5	2025-06-01
102	1	Entwickler	40	2025-02-15

Wie heisst die Beziehungstabelle und welche Spalten hat sie?



Welches Schema ist korrekt?

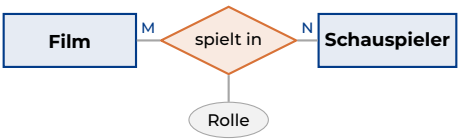
A Film_Schauspieler(#Film_ID, #Schauspieler_ID)

B Film_Schauspieler(#Film_ID, #Schauspieler_ID, Rolle)

C Besetzung(ID, #Film_ID, #Schauspieler_ID, Rolle)

Quiz: M:N-Transformation

Wie heisst die Beziehungstabelle und welche Spalten hat sie?



Welches Schema ist korrekt?

- **A** Film_Schauspieler(#Film_ID, #Schauspieler_ID)
- **B** Film_Schauspieler(#Film_ID, #Schauspieler_ID, Rolle)
- **C** Besetzung(ID, #Film_ID, #Schauspieler_ID, Rolle)

Antwort: B ist korrekt (C wäre auch möglich, aber unnötig)

Hands-on

M:N-Beziehungen transformieren

marimo: 07-er-zu-sql.py

Aufgabe 7.2

Pause

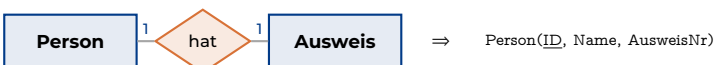
15 Minuten

Agenda

Transformation: 1:1-Beziehung

Drei Optionen:

1. **Zusammenlegen** (wenn sinnvoll)
   Person + Personalausweis → Eine Tabelle
2. **FK auf einer Seite** (bei optionaler Teilnahme)
   FK dorthin, wo die Beziehung **optional** ist
3. **Eigene Beziehungstabelle** (selten)
   Wie bei M:N, aber mit UNIQUE-Constraint



Praxistipp

1:1-Beziehungen sind selten. Oft kann man die Entitäten zusammenlegen.

1:1-Beziehung: Wann FK auf welcher Seite?

Entscheidungskriterium: Optionalität der Teilnahme

Interpretation:

- Jeder Laptop gehört zu **genau einem** Mitarbeiter (total)
- Ein Mitarbeiter hat **höchstens einen** Laptop (partiell)

Lösung: FK auf der **totalen Seite** (Laptop)

Laptop(ID, Modell, #Mitarbeiter_ID NOT NULL)

Regel

FK auf der Seite mit **totaler Teilnahme** – dort ist der FK niemals NULL.

1:1-Beziehung: SQL mit UNIQUE

Problem: Wie verhindert man, dass ein Mitarbeiter zwei Laptops hat?

```sql
CREATE TABLE Mitarbeiter (
    ID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Laptop (
    ID INT PRIMARY KEY,
    Modell VARCHAR(100),
    Mitarbeiter_ID INT UNIQUE,  -- UNIQUE verhindert Duplikate!
    FOREIGN KEY (Mitarbeiter_ID)
        REFERENCES Mitarbeiter(ID)
);
```

Der UNIQUE-Constraint:

- Stellt sicher: Jeder Mitarbeiter_ID-Wert kommt **höchstens einmal** vor
- Zusammen mit FK: 1:1-Beziehung ist gewährleistet

Transformation: Schwache Entitäten

Schwache Entität: Existiert nur in Abhängigkeit von einer anderen.

Besonderheit:

- Der PK der schwachen Entität enthält den FK zum “Owner”
- Zusammengesetzter Primärschlüssel: (Gebäude_ID, RaumNr)
- “Raum 101” ist nur eindeutig mit Gebäude!

Schwache Entität: SQL-Beispiel

```sql
CREATE TABLE Gebaeude (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Adresse VARCHAR(200)
);

CREATE TABLE Raum (
    Gebaeude_ID INT,
    RaumNr VARCHAR(10),          -- z.B. "101", "EG-05"
    Groesse INT,                 -- in qm
    Kapazitaet INT,
    PRIMARY KEY (Gebaeude_ID, RaumNr),
    FOREIGN KEY (Gebaeude_ID)
        REFERENCES Gebaeude(ID)
        ON DELETE CASCADE        -- Räume löschen, wenn Gebäude weg
);
```

Datenbeispiel:

Gebaeude_ID	RaumNr	Groesse	Kapazitaet
1	101	50	30
1	102	25	15

Problem: Mehrwertige Attribute verletzen die 1. Normalform.

Lösung: Auslagern in eigene Tabelle

- Neue Tabelle mit FK zur Hauptentität
- Zusammengesetzter PK: (Person_ID, Telefonnummer)

Mehrwertige Attribute: SQL-Beispiel

DATA DRIVEN DECISIONS

Falsch: Liste in einer Spalte (verletzt Atomarität!)

```
-- FALSCH! Nicht machen!
CREATE TABLE Person (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Telefone VARCHAR(500)  -- "0931-123, 0170-456, ..."
);
```

Richtig: Separate Tabelle

```
CREATE TABLE Person (
    ID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Person_Telefon (
    Person_ID INT,
    Telefonnummer VARCHAR(20),
    Typ VARCHAR(20),              -- 'Mobil', 'Arbeit', etc.
    PRIMARY KEY (Person_ID, Telefonnummer),
    FOREIGN KEY (Person_ID) REFERENCES Person(ID)
);
```

Transformationsregeln: Übersicht

ER-Element	Relationales Modell
Entität	Tabelle
Attribut	Spalte
Schlüsselattribut	Primärschlüssel
1:N-Beziehung	FK auf N-Seite
M:N-Beziehung	Beziehungstabelle mit 2 FKs
1:1-Beziehung	Zusammenlegen oder FK mit UNIQUE
Schwache Entität	FK ist Teil des PK
Mehrwertiges Attribut	Eigene Tabelle
Beziehungsattribut	Spalte in Beziehungstabelle (M:N) oder in Tabelle mit FK (1:N)

Häufige Fehler bei der Transformation

DATA DRIVEN DECISIONS

✗ Häufige Fehler:

1. FK auf der falschen Seite bei 1:N
2. M:N ohne Beziehungstabelle
3. Beziehungsattribute vergessen
4. Listen in einer Spalte speichern
5. FK-Constraints vergessen
6. PK bei schwachen Entitäten falsch

✓ Korrekte Lösung:

1. FK immer auf N-Seite
2. Immer eigene Tabelle erstellen
3. Zur N-Seite oder Beziehungstabelle
4. Eigene Tabelle verwenden
5. FOREIGN KEY ... REFERENCES ...
6. FK + partieller Schlüssel = PK

Checkliste vor dem Abschluss

- Jede Tabelle hat einen Primärschlüssel?
- Alle Beziehungen durch FK abgebildet?
- Keine Listen oder Wiederholungsgruppen?

Agenda

CREATE TABLE: Syntax

```
CREATE TABLE Tabellenname (
    Spalte1 DATENTYP [CONSTRAINT],
    Spalte2 DATENTYP [CONSTRAINT],
    ...
    [PRIMARY KEY (Spalte)],
    [FOREIGN KEY (Spalte) REFERENCES AndereTabelle(Spalte)]
);
```

Wichtige Datentypen:

Typ	Beschreibung	Beispiel
INT	Ganzzahl	42
VARCHAR(n)	Text (max. n Zeichen)	'Müller'
DECIMAL(p,s)	Dezimalzahl (p Stellen, s nach Komma)	19.99
DATE	Datum	'2026-01-28'
BOOLEAN	Wahrheitswert	TRUE / FALSE

Constraint: NOT NULL

Bedeutung: Die Spalte darf **niemals leer** sein.

```
CREATE TABLE Kunde (
    ID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,        -- Pflichtfeld!
    Email VARCHAR(200) NOT NULL,       -- Pflichtfeld!
    Telefon VARCHAR(50)                -- optional (NULL erlaubt)
);
```

Wann verwenden?

- Pflichtfelder bei der Dateneingabe
- Wichtige Geschäftsinformationen
- Fremdschlüssel (bei totaler Teilnahme)

Hinweis

PRIMARY KEY impliziert automatisch NOT NULL!

Constraint: UNIQUE

Bedeutung: Jeder Wert darf **nur einmal** vorkommen.

```
CREATE TABLE Mitarbeiter (
    ID INT PRIMARY KEY,
    Email VARCHAR(200) UNIQUE,          -- keine doppelten Emails
    SVNr CHAR(10) UNIQUE,               -- eindeutige SV-Nummer
    Name VARCHAR(100)                   -- Duplikate erlaubt
);
```

Unterschied zu PRIMARY KEY:

	PRIMARY KEY	UNIQUE
Erlaubt NULL	Nein	Ja (einmal)
Anzahl pro Tabelle	Genau 1	Beliebig viele
Referenzierbar als FK	Ja	Ja

Bedeutung: Werte müssen eine **Bedingung** erfüllen.

```
CREATE TABLE Produkt (
    ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Preis DECIMAL(10,2) CHECK (Preis >= 0),
    Lagerbestand INT CHECK (Lagerbestand >= 0),
    Kategorie VARCHAR(50) CHECK (Kategorie IN
        ('Elektronik', 'Kleidung', 'Lebensmittel'))
);
```

Typische CHECK-Bedingungen:

- Zahlen: CHECK (Wert > 0), CHECK (Alter BETWEEN 0 AND 150)
- Vergleiche: CHECK (Ende > Start)
- Listen: CHECK (Status IN ('aktiv', 'inaktiv', 'gesperrt'))

Constraint: DEFAULT

Bedeutung: Automatischer Wert, wenn keiner angegeben.

```
CREATE TABLE Bestellung (
    ID INT PRIMARY KEY,
    Datum DATE DEFAULT CURRENT_DATE,      -- heute
    Status VARCHAR(20) DEFAULT 'neu',     -- Startwert
    Prioritaet INT DEFAULT 3,             -- mittlere Prio
    Bearbeiter_ID INT DEFAULT NULL        -- noch niemand
);

-- Einfügen ohne Datum und Status
INSERT INTO Bestellung (ID) VALUES (1);
-- Ergebnis: ID=1, Datum=heute, Status='neu', Prioritaet=3
```

Typische DEFAULT-Werte:

- `DEFAULT CURRENT_DATE` – aktuelles Datum
- `DEFAULT CURRENT_TIMESTAMP` – aktueller Zeitstempel
- `DEFAULT 0`, `DEFAULT ''` – Nullwerte

CREATE TABLE: Alle Constraints kombiniert

DATA DRIVEN DECISIONS

```sql
CREATE TABLE Spieler (
    ID INT PRIMARY KEY,                  -- Primärschlüssel
    Name VARCHAR(100) NOT NULL,          -- Pflichtfeld
    Rueckennummer INT UNIQUE,            -- Eindeutig
    Alter INT CHECK (Alter >= 16),       -- Bedingung
    Verein_ID INT DEFAULT 0,             -- Standardwert
    FOREIGN KEY (Verein_ID)              -- Fremdschlüssel
        REFERENCES Verein(ID)
        ON DELETE SET NULL
);
```

Constraint-Typen im Überblick:

- `PRIMARY KEY` – Eindeutig + NOT NULL
- `NOT NULL` – Wert erforderlich
- `UNIQUE` – Keine Duplikate
- `CHECK` – Bedingung prüfen
- `DEFAULT` – Standardwert
- `FOREIGN KEY` – Referenz mit Aktionen

Benannte Constraints

Vorteil: Bessere Fehlermeldungen und einfachere Änderungen

```
CREATE TABLE Mitarbeiter (
    ID INT,
    Name VARCHAR(100),
    Email VARCHAR(200),
    Gehalt DECIMAL(10,2),
    Abteilung_ID INT,

    CONSTRAINT pk_mitarbeiter PRIMARY KEY (ID),
    CONSTRAINT uq_email UNIQUE (Email),
    CONSTRAINT chk_gehalt CHECK (Gehalt > 0),
    CONSTRAINT fk_abteilung FOREIGN KEY (Abteilung_ID)
        REFERENCES Abteilung(ID)
);
```

Fehlermeldung ohne Name: “Constraint violation”
Fehlermeldung mit Name: “Constraint chk_gehalt violated”

Vollständiges Beispiel: Online-Shop

DATA DRIVEN DECISIONS

```sql
CREATE TABLE Kategorie (
    ID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE Produkt (
    ID INT PRIMARY KEY,
    Name VARCHAR(200) NOT NULL,
    Preis DECIMAL(10,2) NOT NULL CHECK (Preis > 0),
    Lagerbestand INT DEFAULT 0 CHECK (Lagerbestand >= 0),
    Kategorie_ID INT NOT NULL,
    FOREIGN KEY (Kategorie_ID) REFERENCES Kategorie(ID)
);

CREATE TABLE Kunde (
    ID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(200) NOT NULL UNIQUE,
    Registriert DATE DEFAULT CURRENT_DATE
);
```

Vollständiges Beispiel: Online-Shop (Forts.)

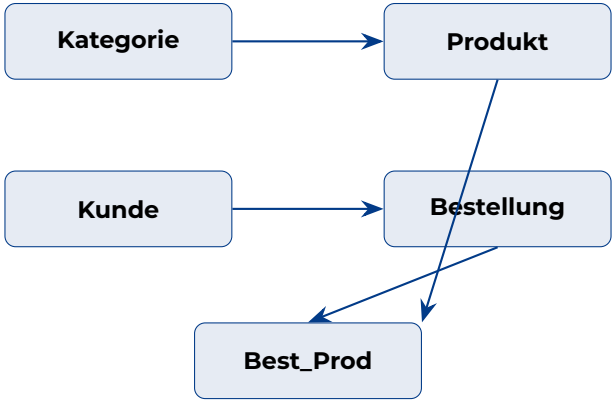
DATA DRIVEN DECISIONS

```
CREATE TABLE Bestellung (
    ID INT PRIMARY KEY,
    Datum DATE DEFAULT CURRENT_DATE,
    Status VARCHAR(20) DEFAULT 'offen'
        CHECK (Status IN ('offen','versendet','storniert')),
    Kunde_ID INT NOT NULL,
    FOREIGN KEY (Kunde_ID) REFERENCES Kunde(ID)
        ON DELETE RESTRICT
);

CREATE TABLE Bestellung_Produkt (
    Bestellung_ID INT,
    Produkt_ID INT,
    Menge INT NOT NULL CHECK (Menge > 0),
    Einzelpreis DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (Bestellung_ID, Produkt_ID),
    FOREIGN KEY (Bestellung_ID) REFERENCES Bestellung(ID),
    FOREIGN KEY (Produkt_ID) REFERENCES Produkt(ID)
);
```

Reihenfolge beim CREATE TABLE

Problem: FK kann nur auf existierende Tabelle verweisen!



Korrekte Reihenfolge:

1. Tabellen **ohne FK** zuerst: Kategorie, Kunde
2. Dann Tabellen **mit FK**: Produkt, Bestellung

Hands-on

ER-Diagramm in SQL umsetzen

marimo: 07-er-zu-sql.py

Aufgaben 7.3 – 7.4

Übung 7.3: Online-Shop

Gegeben (aus Session 6):

- Kunde (1) – gibt auf – Bestellung (N)
- Bestellung (M) – enthält – Produkt (N) [mit Menge]
- Produkt (N) – gehört zu – Kategorie (1)

Aufgabe:
Schreiben Sie die CREATE TABLE-Statements für alle Tabellen.

Hinweise:

- Kunde: ID, Name, Email
- Produkt: ID, Name, Preis, Kategorie_ID
- Bestellung: ID, Datum, Kunde_ID
- Kategorie: ID, Name
- Bestellung_Produkt: Bestellung_ID, Produkt_ID, Menge

Übung 7.4: Bundesliga

ER-Modell:

- Verein (1) – hat – Spieler (N)
- Verein (2) – spielt – Spiel (N) [Heim/Gast]

Aufgabe:

1. Erstellen Sie das relationale Schema
2. Schreiben Sie die CREATE TABLE-Statements
3. Beachten Sie: Spiel hat **zwei** Fremdschlüssel zu Verein!

Hinweis: Die “spielt”-Beziehung ist eine spezielle 2:N-Beziehung.

Schema-Vorschlag

Spiel(ID, Datum, #Heim_ID, #Gast_ID, ToreHeim, ToreGast)

Zusammenfassung

DATA DRIVEN DECISIONS

Relationales Modell:

- Alle Daten in Tabellen
- Zeilen = Tupel
- Spalten = Attribute

Schlüssel:

- Primärschlüssel (PK) – eindeutig
- Fremdschlüssel (FK) – Verweis
- FK-Aktionen: CASCADE, RESTRICT, SET NULL

Transformationsregeln:

1:N	FK auf N-Seite
M:N	Beziehungstabelle
1:1	Zusammenlegen/FK+UNIQUE

SQL-Constraints:

- `PRIMARY KEY`
- `FOREIGN KEY REFERENCES`
- `NOT NULL, UNIQUE`
- `CHECK, DEFAULT`

Der Weg

Reale Welt → ER-Modell → Relationales Schema → SQL

Checkliste: Transformation

DATA DRIVEN DECISIONS

Verwenden Sie diese Checkliste bei jeder Transformation:

- ☐ Jede Entität ist eine Tabelle mit PK
- ☐ Jedes Attribut ist eine Spalte mit passendem Typ
- ☐ 1:N-Beziehungen: FK auf der N-Seite
- ☐ M:N-Beziehungen: Eigene Beziehungstabelle
- ☐ Beziehungsattribute nicht vergessen
- ☐ Schwache Entitäten: FK ist Teil des PK
- ☐ Mehrwertige Attribute: Eigene Tabelle
- ☐ NOT NULL für Pflichtfelder
- ☐ UNIQUE für Alternativschlüssel
- ☐ CHECK für Geschäftsregeln
- ☐ FK-Aktionen (ON DELETE/UPDATE) definiert
- ☐ CREATE TABLE in korrekter Reihenfolge

Vorlesung 8: Normalisierung

- Wann ist ein Schema “gut”?
- Funktionale Abhängigkeiten
- Normalformen: 1NF, 2NF, 3NF, BCNF
- Anomalien systematisch vermeiden

Fragen?

christoph.flath@uni-wuerzburg.de