LU04b - Die Crow’s-Foot-Notation

„Crow’s Foot“ bedeutet übersetzt **Krähenfuss**. Der Name kommt vom Symbol für *many* (viele), das

wie ein kleiner Krähenfuss aussieht.

Die **Crow’s-Foot-Notation** stellt eine Datenbank so dar, dass man sofort erkennt:

- wie **Entitäten** (Tabellen) miteinander verknüpft sind,
- wie viele Objekte (Instanzen) einer Entität mit wie vielen Objekten einer anderen verbunden sein können (= **Kardinalität**),
- welche **Primärschlüssel (PK)** und **Fremdschlüssel (FK)** definiert sind,
- welche **Attribute** zu einer Tabelle gehören (auf Wunsch auch mit Datentypen).

Damit kann die Crow’s-Foot-Notation Teil eines **logischen Datenbankschemas** sein und bildet die Brücke von der Analyse (Chen-Notation) zur technischen Umsetzung in SQL.

Kardinalitäten in der Crow’s-Foot-Notation

Die **Kardinalität** beschreibt die Art der Beziehung zwischen zwei Entitäten. Es gibt drei Haupttypen:

1. **One-to-One (1:1)** Eine Instanz von Entität A gehört genau zu einer Instanz von Entität B.

Beispiel:

- Ein:e Mitarbeitende ↔ genau ein Arbeitsplatz
- Jeder Reisepass ↔ genau eine Person

Darstellung: Linie mit kurzem Strich auf beiden Seiten.

One-to-One (1:1) Beziehung: Ein *Auto kann (gleichzeitig) immer nur* einen *Fahrer,* eine *Fahrerin haben.* Eine *Fahrerin,* ein *Fahrer kann (gleichzeitig) nur immer* Ein *Auto fahren.*

2. **One-to-Many (1:N)** Eine Instanz von Entität A kann mit vielen Instanzen von Entität B verbunden sein. Umgekehrt gehört jede Instanz von B zu genau einer Instanz von A.

Beispiel:

- Ein Regisseur ↔ viele Filme
- Ein Spotify-Artist ↔ viele Songs
- Eine Klasse ↔ viele Lernende

Darstellung: Linie mit Strich (= „eins“) auf Seite A und einem „Krähenfuss“ (drei Linien) auf Seite B.

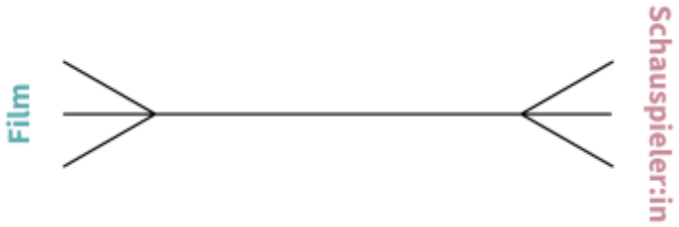
One-to-Many (1:n) Beziehung: Ein *Regisseur,* eine *Regisseurin kann* mehrere *Filme drehen.* Ein *Film wird (in der Regel) nur von* einem *Regisseur,* einer *Regisseurin gedreht.*

3. **Many-to-Many (M:N)** Mehrere Instanzen von Entität A können mit mehreren Instanzen von Entität B verknüpft sein.

Beispiel:

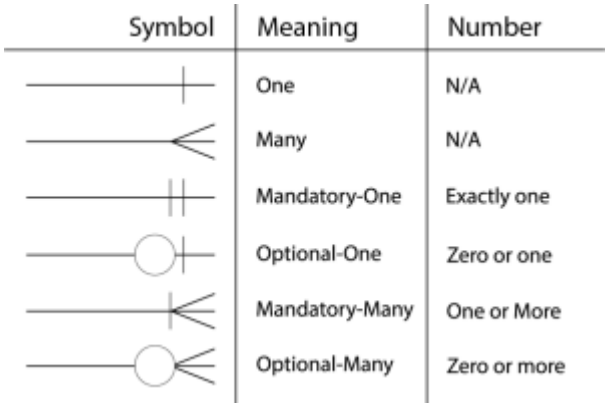
- Mehrere Schauspieler:innen ↔ spielen in mehreren Filmen
- Viele Lernende ↔ besuchen mehrere Module
- Viele Songs ↔ sind in mehreren Playlists enthalten

Darstellung: Krähenfüsse auf beiden Seiten der Beziehung.



Many-to-Many (n:m) Beziehung: Ein:e *Schauspieler:in kann in* mehreren *Filme spielen.* Ein *Film kann* mehrere *Schauspieler:innen haben.*

Die folgende Abbildung zeigt alle möglichen Kardinalitäten:



Symbol	Meaning	Number
	One	N/A
	Many	N/A
	Mandatory-One	Exactly one
	Optional-One	Zero or one
	Mandatory-Many	One or More
	Optional-Many	Zero or more

Warum Crow’s-Foot-Notation?

Die **Crow’s-Foot-Notation** wird oft in der Praxis eingesetzt, weil sie eine **klare und technische Darstellung** bietet.

Im Unterschied zur Chen-Notation, die eher **fachlich und abstrakt** ist, dient Crow’s-Foot als **Brücke zur Umsetzung in SQL**:

- Sie zeigt **Entitäten als Tabellen** mit ihren Attributen.
- Sie markiert **Primärschlüssel (PK)** und **Fremdschlüssel (FK)**.
- Sie stellt **Beziehungen und Kardinalitäten** (1:1, 1:n, n:m) deutlich dar.
- Es können auch die Datentypen der Attribute (Spalten) vermerkt werden.

Damit ist sie die **letzte Station vor der praktischen Umsetzung** mit SQL-Befehlen wie *CREATE TABLE ….*

Crow’s-Foot wird also vor allem von **Entwickler:innen, Datenbank-Designer:innen und Lernenden** genutzt, die den Schritt von der Analyse (Chen) zur technischen Umsetzung (SQL) machen.

Beispiel Crow’s-Foot-Notation

Stellen wir uns einen **Online-Buchshop** vor. Wir haben die Entitäten:

- **Kunde**
- **Bestellung**
- **Produkt**

Beziehungen:

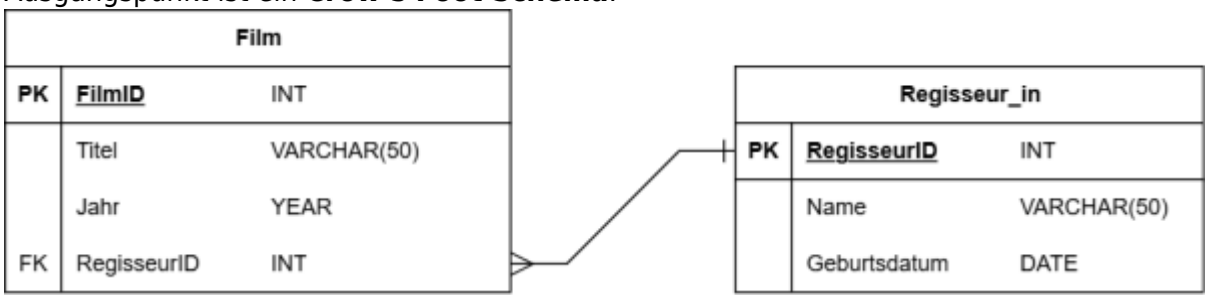
- Ein Kunde kann viele Bestellungen aufgeben (1:N).
- Eine Bestellung kann mehrere Produkte enthalten, und ein Produkt kann in vielen Bestellungen vorkommen (M:N).

Darstellung in Crow's-Foot-Notation:

- Linie zwischen Kunde und Bestellung, mit Strich (1) auf der Kundenseite und Krähenfuss (N) auf der Bestellungsseite.
- Linie zwischen Bestellung und Produkt mit Krähenfuss auf beiden Seiten.

Beispiel: Vom Schema zur Tabelle

Ausgangspunkt ist ein **Crow's Foot Schema**:



- **Regisseur** (RegisseurID **PK**, Name)
- **Film** (FilmID **PK**, Titel, Jahr, RegisseurID **FK**)
- Beziehung: **1 Regisseur ↔ n Filme**

Umsetzung in SQL

1. Datenbank anlegen

```
CREATE DATABASE filmdatenbank;
USE filmdatenbank;
```

2. Tabelle „regisseur“ anlegen

```
CREATE TABLE regisseur (
  regisseur_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  birthday DATE
);
```

3. Tabelle „film“ anlegen

```
CREATE TABLE film (
  film_id INT AUTO_INCREMENT,
  titel VARCHAR(50) NOT NULL,
  jahr YEAR,
  regisseur_id INT,
  PRIMARY KEY (film_id),
  FOREIGN KEY (regisseur_id) REFERENCES
regisseur(regisseur_id)
);
```

Sowohl das setzen von Foreign Keys (Fremdschlüsseln) in SQL, wie auch das gleichzeitige Abfragen von Daten aus zwei Tabellen (s. unten) werden wir zu einem späteren Zeitpunkt im Modul behandeln. Das dient hier dafür, dass Sie einen Überblick bekommen, wie es gehen würde. Das (also das setzen von Fremdschlüsseln und das programmieren von verschachtelten Abfragen) ist nicht Teil der Leistungsbeurteilung 1.

4. Daten einfügen

Fügen Sie mit Webstorm manuell (Tabelle anklicken und dann → „Edit Data“ (Tabellen-Icon)) ein paar Daten in die beiden Tabellen. Sie können zur Inspiration Daten aus der Tabelle *imbd_top_1000* verwenden, welche wir zu einem früheren Zeitpunkt bereits importiert haben.

5. Daten abfragen

Mit einem Select können wir beide Tabellen gleichzeitig abfragen:

```
SELECT f.titel, f.jahr, r.name AS regisseur
FROM film f, regisseur r
WHERE f.regisseur_id = r.regisseur_id;
```

Was zu einem ähnlichen Ergebnis, wie hier führen könnte:

Titel	Jahr	Regisseur
Inception	2010	Christopher Nolan
Lost in Translation	2003	Sofia Coppola