Datenbank-Design

Das Relationale Datenmodell wurde entwickelt, um Schwächen (sog. „Anomalien“) der bis dahin gebräuchlichen Datenmodelle zu beheben.

Diese Anomalien machen sich bemerkbar bei

- Update (Aktualisieren von Daten)
- Delete (Löschen von Daten)
- Insert (Einfügen von Daten)

Das Verfahren, mit dem man diese Anomalien beseitigt, heißt **Normalisierung**.

Die einzelnen Schritte der Normalisierung nennt man **Normalformen**.

Es gibt eine 1. bis 3. Normalform, eine Boyce-Codd-Normalform und eine 3. bis 5. Normalform, und jede Normalform hat ganz präzise Regeln. (in der Praxis ist aber vieles Ermessenssache)

Die Regeln für die erste bis dritte Normalform:

1. Normalform

Eine Tabelle ist in der ersten Normalform

- wenn der Wert in einem jeden Feld atomar ist. (Aus diesem Grund kann man den MySQL-Datentyp SET kritisch sehen, weil ein solches Feld ja ein Array als Wert erlaubt)
- wenn für jede Tabelle ein Schlüssel definiert ist, der jeden Datensatz eindeutig definiert. (Diese Bedingung ist immer erfüllt, wenn die Tabelle einen Primärschlüssel hat.
- wenn es keine Wiederholungsspalten (telefon_1, telefon_2) gibt
- wenn keine identischen Datensätze existieren
- wenn es keine sich wiederholenden Gruppen von Daten gibt.

2. Normalform

Schon die zweite Normalform wird nur nötig, wenn der Schlüssel in der ersten Normalform aus mehrern Spalten zusammengesetzt ist.

Die zweite Normalform ist erfüllt

- wenn jede Nicht-Schlüsselspalte vom ganzen Schlüssel abhängig ist und nicht nur von einem Teil des Schlüssels.

3.Normalform

Die dritte Normalform ist erfüllt, wenn es keine sogenannten **transitiven Abhängigkeiten** in einer Tabelle gibt.

Transitive Abhängigkeit bedeutet, dass ein Attribut indirekt vom Schlüssel abhängig ist, also von einem Attribut abhängig ist, das vom Schlüssel abhängig ist.

Beziehungen zwischen Tabellen

In der Praxis erreicht man Normalisierung, indem man aus denjenigen Attribute einer Tabelle, die die jeweilige Normalform verletzen, eine eigene Tabelle macht, solange bis alle Regeln der Normalisierung erfüllt sind.

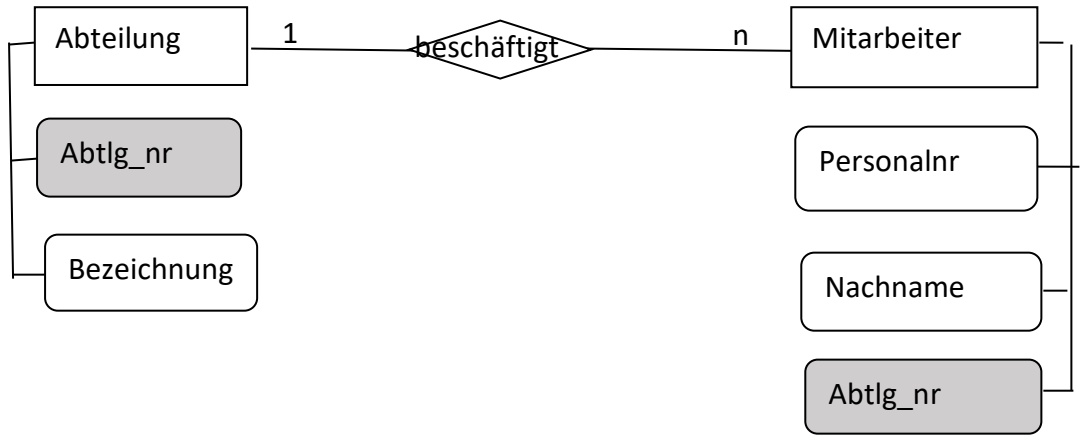
Dann muss man natürlich die zusammengehörenden Informationen auch wieder zusammensetzen können. Dazu fügt man in der zweiten Tabelle ein sog. Feld ein, in dem die Primärschlüsselwerte der ersten Tabelle als Fremdschlüssel geführt werden.

Die Beziehungen zwischen Tabellen bildet man in **Entity-Relationship-Modellen** ab.

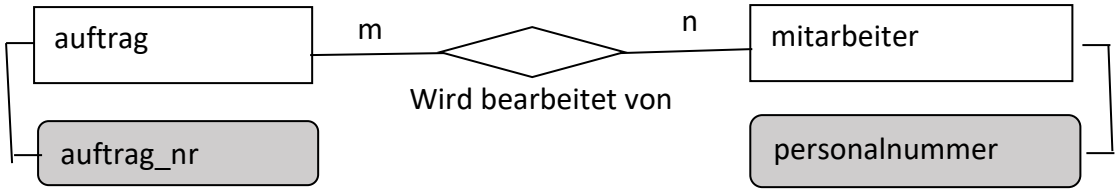
Der häufigste Beziehungstyp ist eine **1:n-Beziehung**:

Einem Datensatz in der ersten Tabelle (**Mastertabelle**)können mehrere Datensätze in der zweiten Tabelle (**Detailtabelle**) zugeordnet sein. Einem Datensatz in der zweiten Tabelle kann aber immer nur ein Datensatz in der ersten Tabelle zugeordnet sein.

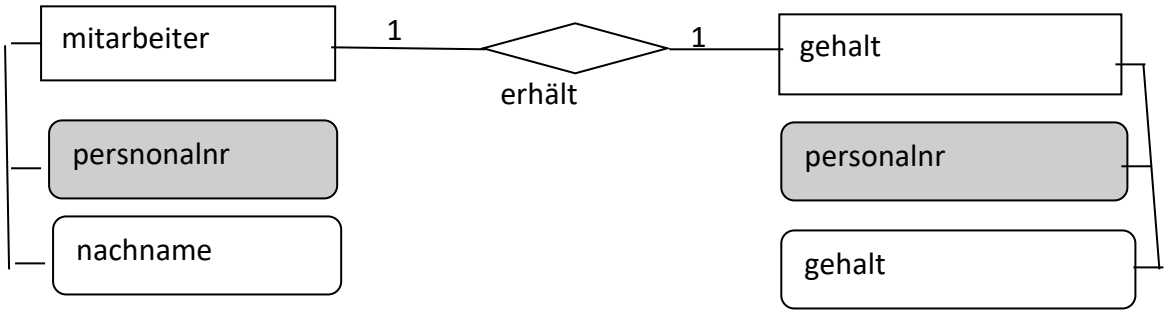
Zum Beispiel können viele Mitarbeiter in einer Abteilung beschäftigt sein, aber ein Mitarbeiter kann immer nur in einer Abteilung arbeiten:



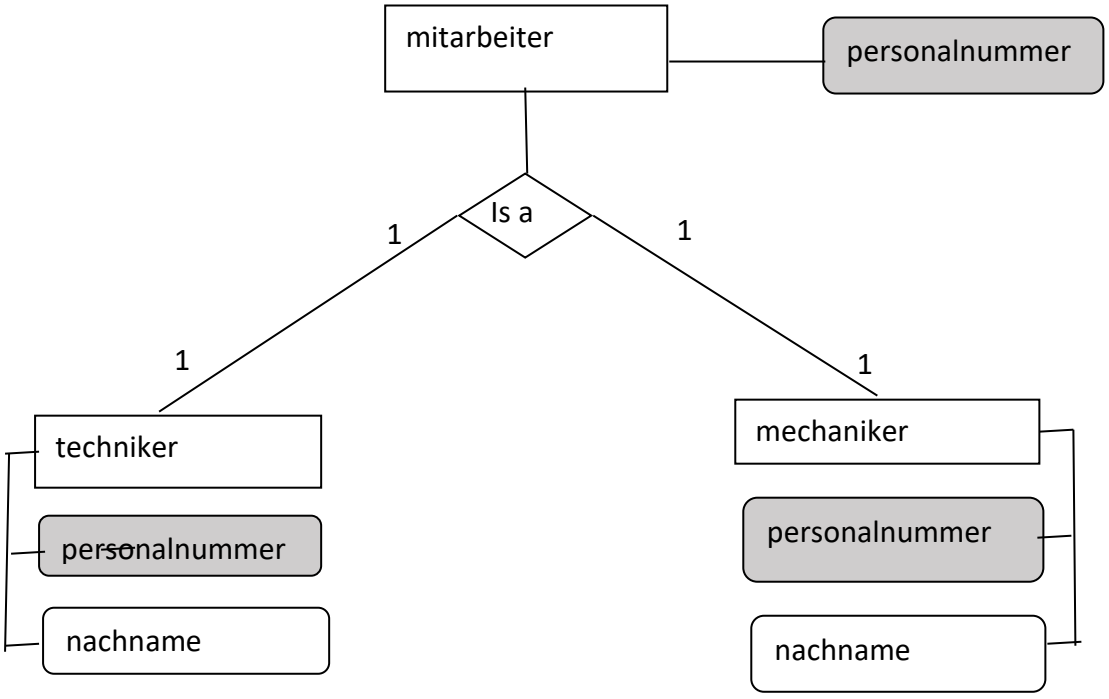
In einer **n:m-Beziehung** können jedem Datensatz in der ersten Tabelle mehrere Datensätze in der zweiten Tabelle zugeordnet sein, aber auch umgekehrt. Praktisch kann man das nur mit einer dritten Tabelle realisieren, die mindestens zwei Attribute hat: jedes der Attribute ist dann ein Fremdschlüsselfeld und die beiden Attribute zusammen bilden einen zusammengesetzten Primärschlüssel. Eine m:n-Beziehung wird also realisiert durch zwei 1:n-Beziehungen mit einer dritten Tabelle.



Bei einer **1:1-Beziehung** entspricht ein Datensatz in der ersten Tabelle einem Datensatz in der zweiten Tabelle und umgekehrt. Das ist in der Praxis nur nötig, wenn eine Tabelle viele Attribute hat oder für die Informationen in einer Tabelle verschiedene Nutzer unterschiedliche Rechte haben.



Bei einer **Is-A-Beziehung** wird der Primärschlüssel vererbt. („Is-A“ ist ein Begriff aus der Objektorientierten Programmierung, mit der die Beziehung einer abgeleiteten Klasse zur Parentklasse bezeichnet wird)



Zu guter Letzt gibt es noch eine Beziehung, bei der der Primärschlüssel in einer Tabelle in derselben Tabelle zum Fremdschlüssel wird: eine **rekursive Beziehung** bzw. **reflexive Beziehung**. (Diese Art von Beziehung ist in MySQL nur mit einem Umweg zu realisieren.) Benötigt wird sie, wenn man Sachverhalte abbilden will wie z. B.: **Wer ist Vorgesetzter von wem?** Oder **Welcher Beitrag ist eine Antwort auf welchen anderen Beitrag?**