Zusammenfassung M153

# Lernziele

* Generalisierung/Spezialisierung
  + disjunkt / nicht disjunkt / partiell / total
* Rekursion
  + Hierarchisch / Netzwerkartig
* ERM

[Lernziele](#_x6vfoq25uai3)

[Generalisierung / Spezialisierung](#_hrzdap5e0lem)

[Grundlegende Erklärung](#_xg8rrp9nde98)

[Disjunkt / Nicht disjunkt](#_23ztsh5xs4wb)

[Beispiel Total Disjunkt](#_klcxlr55qwik)

[Beispiel Partiell Disjunkt](#_pap2581dtgv)

[Beispiel Total Nicht-Disjunkt](#_28id90jv684g)

[Beispiel Partiell Nicht-Disjunkt](#_vtl4rz9apqcu)

[Rekursion](#_x1ze6wgkifgt)

[Grundlegende Aufklärung](#_opmmaiuyrpac)

[Hierarchische Rekursion](#_glzorjot977e)

[Netzwerkartige Rekursion](#_dt5gomxiecg0)

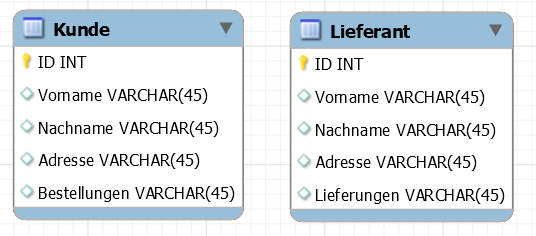
[ERM](#_s23gd6ra1guq)

## 

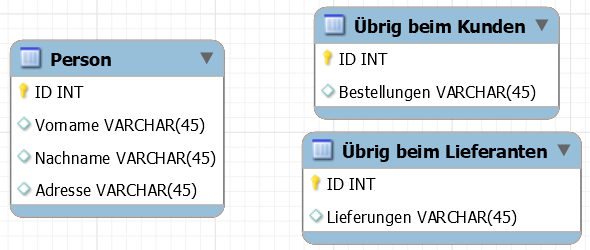
## Generalisierung / Spezialisierung

### Grundlegende Erklärung

Die Generalisierung beschreibt den Vorgang, alle **gemeinsamen** Eigenschaften von unterschiedlichen Entitäten in einer Tabelle zusammenzuführen.



Wenn wir nun beispielsweise Händler sind und unsere Kunden sowie Lieferanten abspeichern müssen, hätten wir folgende 2 Tabellen:



Wenn wir diese nun generalisieren erhalten wir eine “generalisierte Tabelle” und das was übrig bleibt:

Damit haben wir auch gleich die **Spezialisierung** vorgenommen, denn die 2 übrigen Tabellen enthalten nur Eigenschaften, welche sich auf den Kunden oder Lieferanten **spezialisieren**/beschränken. Für unsere Datenbank stehen diese Tabellen jedoch noch in keiner Beziehung zueinander. Sie weiss also nicht, ob eine Person ein Kunde oder Lieferant ist. Für diese Beziehungen müssen wir entscheiden, ob sie disjunkt oder nicht disjunkt sind. Grundsätzlich braucht es für jede Art von Beziehung aber Fremdschlüssel in den beiden spezialisierten Tabellen (Siehe folgende Beispiele).

### Disjunkt / Nicht disjunkt

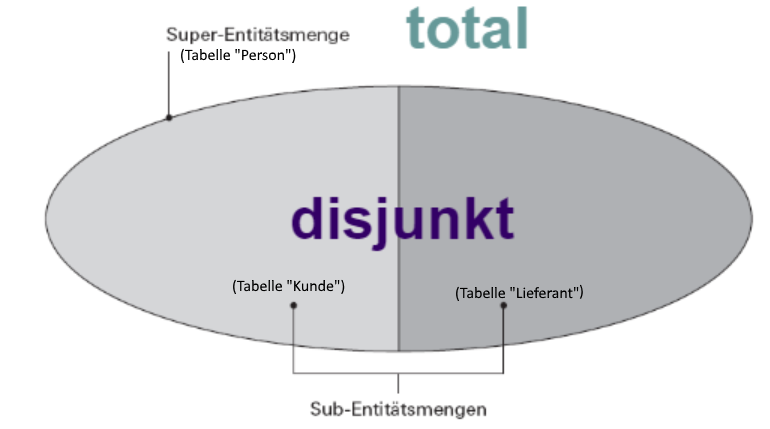
Ab jetzt sprechen wir von einer Super-Entitätsmenge “Person” (Unsere Generalisierte Tabelle) und 2 Sub-Entitätsmengen “Lieferant” und “Kunde” (Die übrig gebliebenen Tabellen). Diese Sub-Entitätsmengen befinden sich ja sozusagen in der Super-Entitätsmenge, da sie aus dieser spezialisiert/ausgelagert wurden.

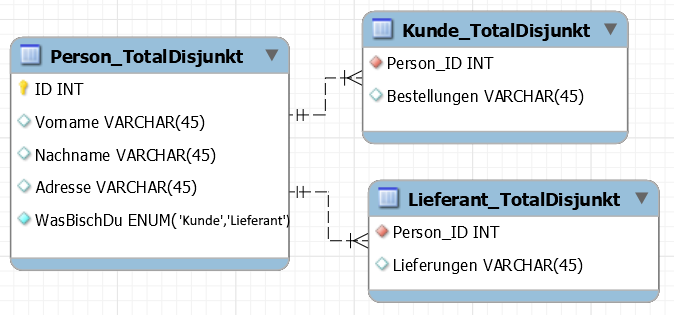
Disjunkt (lateinisch disjunctus (-a, -um) = getrennt) und Nicht-disjunkt beschreibt die Verbindung, welche die Sub-Entitätsmengen zu einander in der Super-Entitätsmenge haben.

#### Beispiel Total Disjunkt

Es ist total offensichtlich, dass eine Person **entweder** ein Kunde **oder** ein Lieferant ist. Es kann auch **nicht** sein, dass eine Person keins von beiden ist, denn dann würden wir sie nicht in der Datenbank wollen.

Die Beziehung zwischen den 2 **Sub-Entitätsmengen** “Lieferant” und “Kunde” ist also **absolut** getrennt, was in der Fachsprache “**Total Disjunkt**” genannt wird.





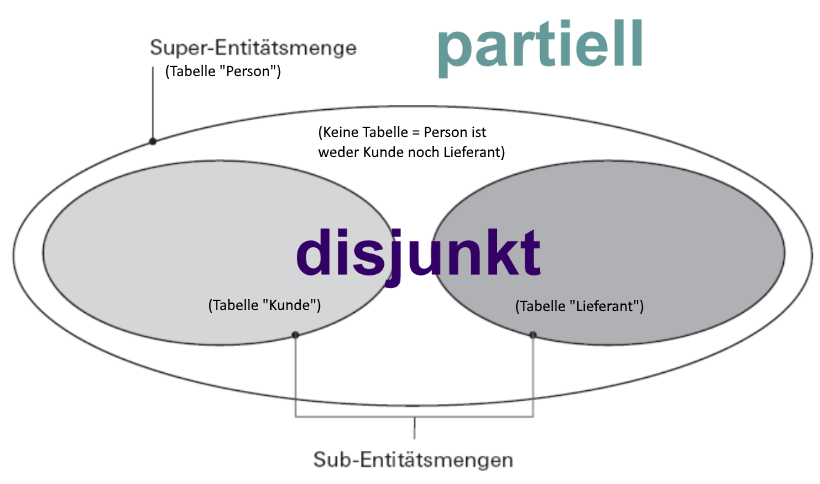
Um dies nun in unserer Datenbank klarzustellen, brauchen wir in der Tabelle “Person” eine Eigenschaft, welche klar aussagt, dass die Person “Kunde” oder “Lieferant” ist. Anscheinend ist die beste Lösung einen **ENUM** zu verwenden. Da eine Person eins von beiden sein **MUSS**, darf dieser ENUM **nicht “Null”** sein:

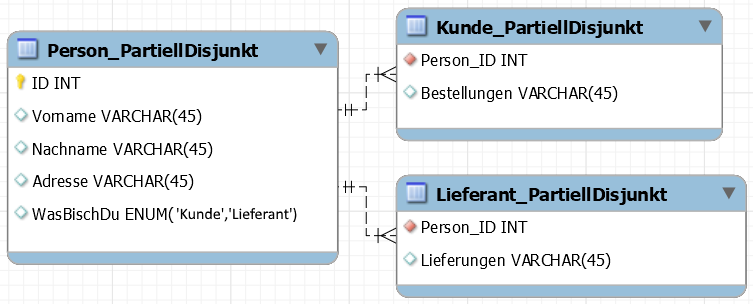
(Es wurden noch Fremdschlüssel in den beiden anderen Tabellen hinzugefügt)

#### 

#### Beispiel Partiell Disjunkt

Wenn wir uns jetzt entscheiden, dass wir auch Personen abspeichern wollen, welche **weder** Kunde **noch** Lieferant sind, benötigen wir eine andere Beziehung. Es ist nun also so, dass eine Person **entweder nichts** ist **oder**, falls sie irgendetwas sein will, **entweder** ein Kunde **ODER** ein Lieferant. Sie kann also noch immer **nicht beides** sein. Die beiden Sub-Entitätsmengen bleiben also **nach wie vor Disjunkt**, da es aber auch Personen gibt welcher keiner der beiden zugeteilt sind ist die Beziehung **nicht** mehr **TOTAL**, sondern nur noch **PARTIELL**.



Damit wir nun einer Person in unserer Datenbank keinen Typ zuweisen müssen, müssen wir einfach das “**NOT NULL**” Attribut des **ENUM’s** aus dem vorherigen Beispiel **entfernen**. Dadurch können wir die die auswahl des ENUM’s einfach weglassen und haben dadurch unsere nicht-zugewiesene Person:

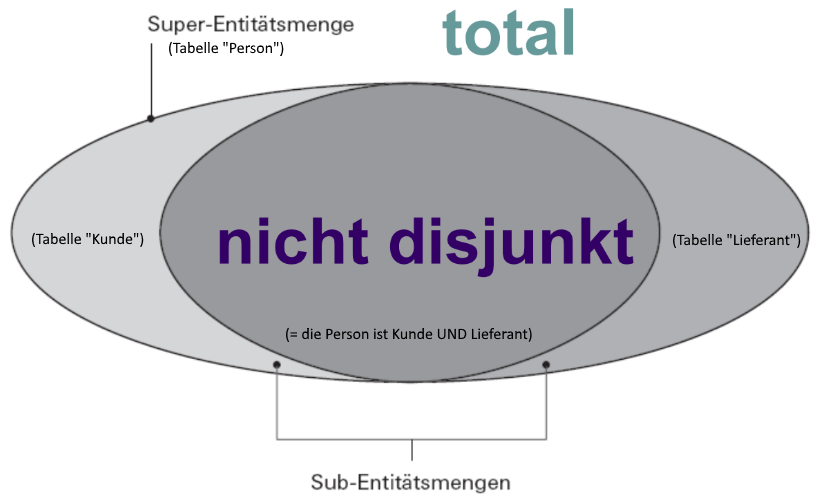
#### 

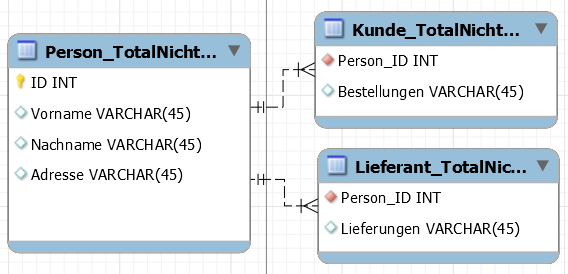
#### Beispiel Total Nicht-Disjunkt

Nehmen wir an, dass einer unserer Lieferanten Computer herstellt und an uns liefert. Diese Computer enthalten Prozessoren, welche der Lieferant irgendwo bestellen muss. Und **wir** verkaufen genau diese Prozessoren zufällig an unsere Kunden.

Der Lieferant möchte nun also seine Prozessoren bei uns Bestellen, was ihn zum Kunden macht. Da wir aber durch die Beispiele zuvor nur mit disjunkten Beziehungen zwischen Lieferanten und Kunden arbeiten, schmiert das ganze System ab als wir den Lieferanten zusätzlich zum Kunden machen wollen (Wir können im **ENUM nicht beide Optionen** auswählen).

Wir legen also neu fest, dass eine Person **entweder** Kunde **oder** Lieferant sein muss, **oder beides**. Wir lassen auch die Option **weg**, dass eine Person **gar nichts** sein kann (aus dem Beispiel “partiell Disjunkt”) da uns das einfach zu bescheuert vorkommt.



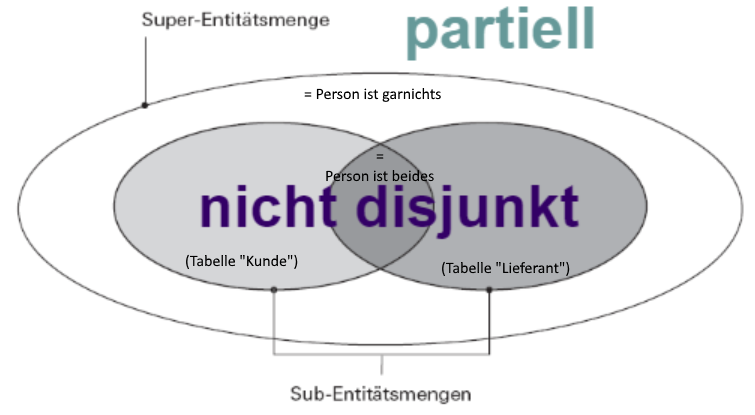
Aber wie sollen wir das in der Datenbank abbilden? Laut der *besten* IT-Schule ist die *beste* und richtige Lösung ganz einfach: Wir **lassen** einfach den **ENUM** aus der Tabelle “Person” **weg**:

Nun sind wir zwar wieder da wo wir angefangen haben, aber egal. Die Beziehungen der spezialisierten Tabellen lassen sich ja noch zur Tabelle “Person” nachvollziehen.

#### Beispiel Partiell Nicht-Disjunkt

Der Begriff **Partiell** bedeutet bei einer **nicht-disjunkten** Tabelle **genau das gleiche** wie bei einer **Disjunkten** Tabelle: Eine Person (Super-Entitätsmenge) **kann auch gar nichts sein**.

Wir wollen nun also auch wieder Personen abspeichern, welche **weder** Kunde **noch** Lieferant sind, und **wenn** sie etwas sind, können sie **entweder** Kunde **oder** Lieferant sein, **oder beides**.



Wie sollen wir das nun in der Datenbank abbilden? Die *beste* Schule sagt wieder: Genau **gleich wie** bei einer “**Totalen** **Nicht-Disjunkten**” Tabelle. Es ändert sich also genau nichts zum vorherigen Beispiel.

Aber wieso? → Da es zuvor schon keine Eigenschaft in der Super-Entitätsmenge (Tabelle Person) gab, welche die Zuweisung zu einer oder beiden Sub-Entitäten festgelegt hat. Durch die **entfernung** des **ENUM’s** aus den beispielen der **Disjunkten Tabellen**, gibt es nun **nichts mehr**, was durch ein “NOT NULL”-Attribut **das zuweisen** zu einer Sub-Entität **notwendig macht**. In diesem Sinne ist eine “Totale Nicht-Disjunkte Tabelle” eigentlich genau das gleiche wie eine “Partiell Nicht-Disjunkte Tabelle”. **Have fun :)**

## 

## Rekursion

### Grundlegende Aufklärung

Rekursion (lateinisch recurrere ‚zurücklaufen‘) bzw. rekursiv beschreibt bei Datenbanken den Zustand, wenn Eigenschaften **einer Tabelle auf dieselbe Tabelle** verweisen. Dies kann z.B. durch Fremdschlüssel der Fall sein.

Es wird also eine **Eigenschaft** in einer Tabelle **benötigt**, **welche** sich sozusagen **bereits in** dieser **Tabelle** befindet.

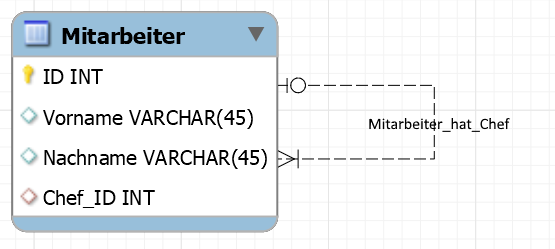
Beispiel: Unser Laden aus dem vorherigen Beispiel hat zahlreiche Mitarbeiter, welche in der Tabelle “Mitarbeiter” abgespeichert werden. Jeder Mitarbeiter hat einen übergeordneten Chef, welcher ebenfalls Mitarbeiter ist und deshalb in der gleichen Tabelle gespeichert wird. Wie sollen wir nun definieren können, welcher Mitarbeiter welchen Mitarbeiter als Chef hat? → **durch Rekursion :)**

### Hierarchische Rekursion

Wenn nun jeder Mitarbeiter **genau einen einzigen** Chef hat, spricht man von einer **hierarchie**. Es gibt also keinen Mitarbeiter welcher mehrere Chefs hat und keinen Chef, der sich einen Mitarbeiter mit einem andere Chef teilt.

Es entsteht also eine **1:n Beziehung** zwischen **Mitarbeiter** und **Chef**

(Mitarbeiter hat 1 Chef, Chef hat n Mitarbeiter).

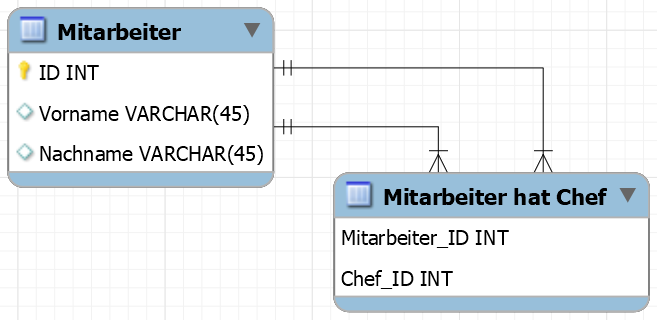
Diese Beziehung können wir **durch eine Rekursion** auf die eigene Tabelle **darstellen**, indem wir jedem Mitarbeiter eine Eigenschaft “Chef” bzw. “Chef\_ID” geben. In dieser Eigenschaft steht dann jeweils die ID des Mitarbeiters, welcher Chef von dem betroffenen Mitarbeiter ist. 

Zu beachten ist, dass der Fremdschlüssel “Chef\_ID” **nicht** das Attribut **“NOT NULL”** haben darf. Es **muss** also **möglich sein** einen Mitarbeiter abzuspeichern, welcher keinen Chef hat. Dieser ist dann sozusagen der *BigBoss* unserer Firma.

### Netzwerkartige Rekursion

Da die **hierarchie** ein veraltetes Konzept ist, entscheiden wir uns in unserer Firma für ein **netzwerkartiges** Konzept, was uns erlaubt **einem Mitarbeiter mehrere Chefs** zuzuweisen. Zum Beispiel hat ein Lehrling einen Fachvorgesetzten **und** einen Vorgesetzten in der Personalabteilung.

Da wir **nicht wissen**, **wie viele** Vorgesetzte ein Mitarbeiter haben könnte, macht es **keinen Sinn** nun **viele** “Chef\_ID” **Fremdschlüssel** wie im vorherigen Beispiel **zu erstellen**. Durch diese **n:m Beziehung** zwischen Mitarbeitenden und Cheffes benötigen wir eine **neue Tabelle**, in welcher diese Beziehungen gespeichert werden.

Die neue Tabelle hat nun also einen Fremdschlüssel “Mitarbeiter\_ID” und einen “Chef\_ID”. Ein Eintrag in dieser Tabelle zeigt also wer der Chef dieses Mitarbeiters ist. 

Diese zwei Fremdschlüssel **können** nun im Gegensatz zum vorherigen Beispiel **auch** die Attribute **“NOT NULL”** enthalten, da es ja keinen Sinn macht hier einen Eintrag für den obersten Cheffe “BigBoss” zu erstellen (Die Chef\_ID Eigenschaft wäre leer bringt also keinem was).

## ERM

Es steht nirgends genau beschrieben, was hier die Lernziele sind. In GDrive befindet sich lediglich ein Ordner “Repetition ERM” also soll man wahrscheinlich einfach wissen wie mach Sachen in MySQL Workbench abbildet. Dies beinhaltet:

* Die verwendung sinnvoller **Datentypen**
* Den **korrekten Einsatz** von **Primär-** und **Fremdschlüsseln**
* Die Einhaltung der Bedingungen bis zur **3. Normalform**

**Erklärungen zu den Normalformen** sind hier gut beschrieben:

* [1. Normalform: Hier](http://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/erste-normalform/)
* [2. Normalform: Hier](http://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/zweite-normalform/)
* [3. Normalform: Hier](http://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/dritte-normalform/)